

1 Aufgabenstellung:

Ein Standard-Industrieroboter vom Typ UNIMATION PUMA 562 mit angeschlossener Steuerung vom Typ UNIMATION MARK 3 soll mittels einer Steuerkugel mit eingebautem Kraft-Momenten-Sensor (Robot-Command-Teach-Ball) gesteuert werden. Ermöglicht wird dieses durch eine Schnittstelle der Steuerung, die den Anschluß eines Fremdrechners erlaubt.

Gleichzeitig soll die Bewegung des Roboters auf einer Grafik-Workstation in Echtzeit dargestellt werden und so bei entfernt aufgestellter Workstation mit Steuerkugel eine Teleoperation ermöglichen. Bei der Workstation handelt es sich um einen Rechner mit Hochleistungsgrafik, auf dem eine Applikation zur Simulation mechanischer Systeme (SMS, Siemens AG) lauffähig ist. In dieser Applikation ist die bezeichnete Fertigungszelle als Simulation hinterlegt. Die Workstation ist mit der Echtzeitdarstellung der Simulation derart belastet, daß zur Kommunikation mit der Robotersteuerung ein externer Kommunikationsrechner erforderlich ist. Eine erste Implementation dieses Ansatzes wurde durch den Einsatz eines Industriestandard-PC's realisiert [DICKEN 92]; die zukünftige Integration eines Bildverarbeitungssystems, das ebenfalls mit der Robotersteuerung kommunizieren soll, macht eine Portierung der Kommunikationsroutinen auf das Echtzeitbetriebssystem OS-9 erforderlich.

2 Aufbau der Fertigungszelle:

Der Gesamtaufbau stellt sich folgendermaßen dar:

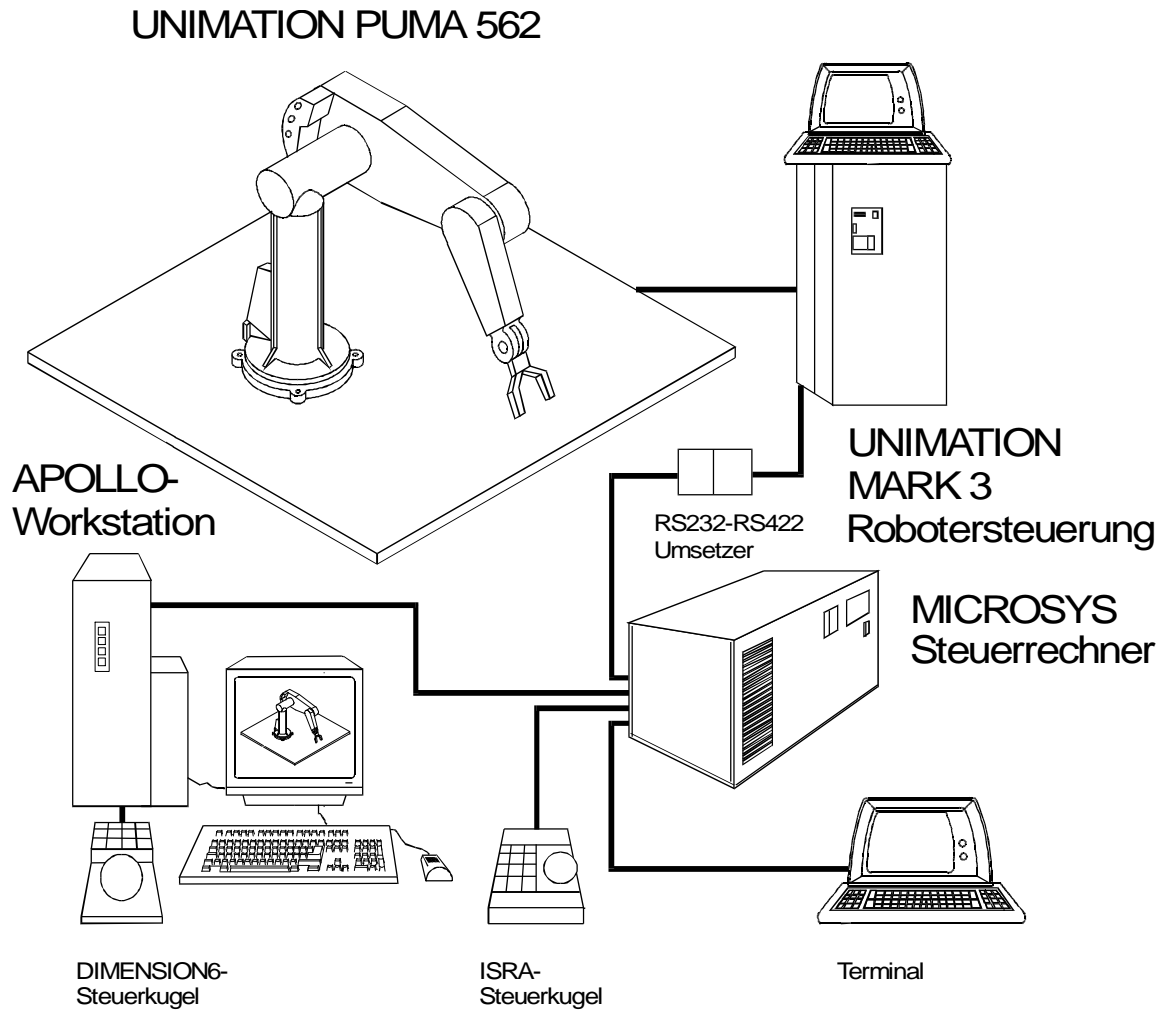


Bild 1. Aufbau

2.1 Roboter und Robotersteuerung

Bei dem UNIMATION Industrieroboter vom Typ PUMA 562 in der Fertigungszelle handelt es sich um einen Kombi-Arm-Roboter, der sechs Freiheitsgrade in der Bewegung aufweist.

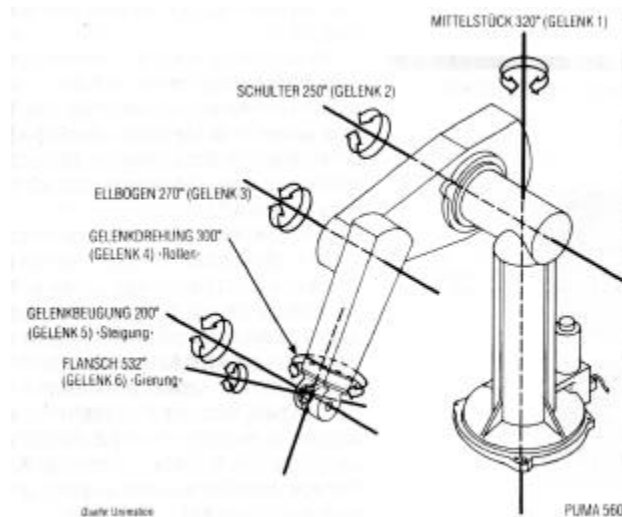


Bild 2. PUMA 562 (Quelle: UNIMATION)

Gelenk 1 schwingt um eine Achse, welche vertikal durch den Rumpf des Roboters verläuft. Gelenk 2 rotiert um eine Achse horizontal durch das Schulterstück und verläuft in einer lotrechten Bewegung zum Rumpf. Das dritte Gelenk rotiert um eine Achse, die der Achse zu Gelenk 2 parallel verläuft. Diese Bewegungen geben dem Arm drei Freiheitsgrade. Hinzu kommen die drei Freiheitsgrade der drei Einzelgelenke des Handgelenks. Die Bewegungen der Gelenke 4, 5 und 6 werden als "Rollen", "Steigung" und "Gierung" bezeichnet.

Die angeschlossene Robotersteuerung UNIMATION MARK3 enthält die Regler für die einzelnen Roboterachsen und die Spannungsversorgung für die Gelenkmotoren. Außerdem ist ein Steuerrechner enthalten, welcher die Programmierung der Roboterbewegungen in der Programmiersprache VAL-II ermöglicht. Die Steuerung erlaubt den Anschluß eines Terminals zur Ein- und Ausgabe der Programme, eines Floppy-Disk-Laufwerkes zur externen Datenspeicherung, eines Teach-Zusatzgerätes zur Handsteuerung des Roboters und den Anschluß eines externen Rechners über eine serielle RS422-Schnittstelle.

2.2 Grafik-Workstation

Bei der eingesetzten Workstation handelt es sich um eine HP-APOLLO vom Typ 725t mit 68040 CPU, 33 MHz Taktfrequenz und einem angeschlossenen Grafikbeschleuniger-Subsystem. Die Workstation läuft unter dem Betriebssystem Domain-OS 10.3.5, das unter anderem Entwicklungswerkzeuge für C- und PASCAL-Programmierung bereitstellt. Die auf dieser Workstation lauffähige Visualisierungssoftware SMS¹ der Firma

¹ SMS: Simulation mechanischer Systeme

SIEMENS erlaubt die 3-dimensionale Darstellung und Simulation von Industrierobotern. Unter SMS steht eine Simulationsumgebung für den verwendeten PUMA 560 Industrieroboter in der genannten Fertigungszelle zur Verfügung. SMS ermöglicht den Anschluß einer Steuerkugel vom Typ DIMENSION6 mit eingebauten Kräfte-/Momenten-Sensoren für sechs Freiheitsgrade. Über diese Steuerkugel ist es möglich, unter SMS die Ansicht auf die simulierte Fertigungszelle komfortabel zu verändern. Es steht eine Programmierschnittstelle in SMS zur Verfügung, die Eingriffe in den Ablauf der Simulation ermöglicht. Diese Programmierschnittstelle erlaubt prinzipiell auch die Kommunikation mit Fremdrechnern über serielle Schnittstellen.

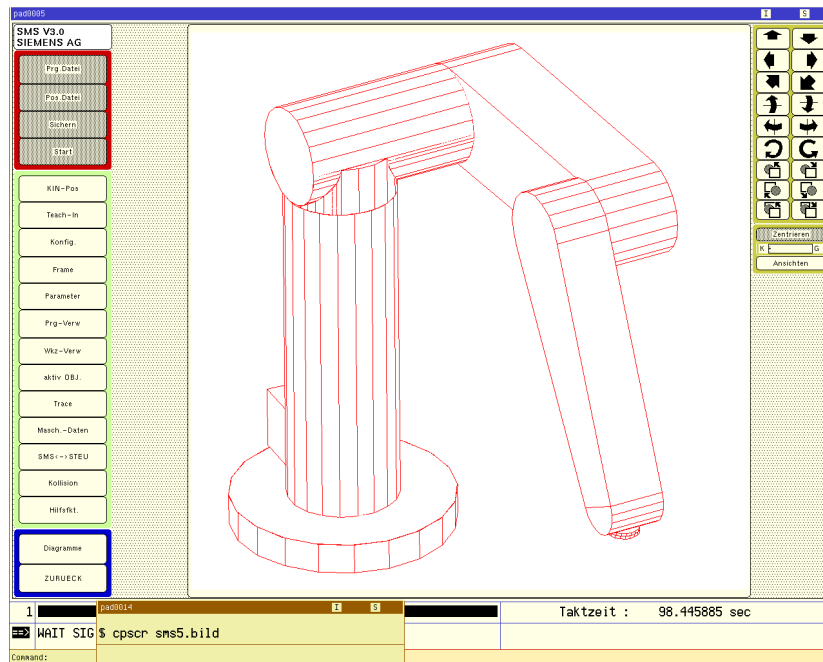


Bild 3. SMS-Oberfläche (Quelle: SIEMENS)

2.3 VME-Bus Bildverarbeitungssystem unter OS-9

Bei dem zur Verfügung stehenden VME-Bus-System handelt es sich um eine CPU-Karte der Firma MICROSYS mit 68040 CPU mit 33 MHz Taktfrequenz. Das System verfügt über eine VME-Bus-Ethernetkarte, eine MICROSYS GRC06 Grafikkarte, diverse Bildverarbeitungskarten, serielle und SCSI-Schnittstellen für den Anschluß externer Peripheriegeräte.

2.4 Steuerkugel ISRA-Robot-Command-Teach-Ball

Der ISRA-Robot-Command-Teach-Ball ist ein 3-Dimensionales Steuergerät für Anwendungen der Computergrafik. Er besteht aus einer Sensorkugel, die elastisch über Federn auf einem Gehäuse angebracht ist. Das Gehäuse enthält neben Spannungsversorgung und serieller RS232-Schnittstelle eine auf einem Microcontroller basierte Auswertelektronik mit integrierter

Folientastatur. Die Steuerung dreidimensionaler Objekte am Bildschirm oder auch in der Realität erfordert eine Kontrolle von sechs Freiheitsgraden (je drei für Translation und Rotation). Diese Forderung erfüllen in die Steuerkugel integrierte optisch arbeitende Kraft- und Momentsensoren. Im Gegensatz zu herkömmlichen Geräten wie Joystick, Tablett, Maus oder Trackball lassen sich hier mit einer Hand ohne Umschalten sechs Freiheitsgrade gleichzeitig kontrollieren.

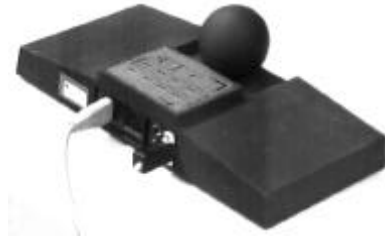


Bild 4. ISRA Robot-Command-Teach-Ball

3 Kommunikation der Robotersteuerung mit einem externen Steuerrechner

Die auf der UNIMATION-Robotersteuerung verfügbare Roboterprogrammiersprache VAL-II ermöglicht neben der Programmierung von Bahnen, Orientierung bzw. Lage des am Roboterflansch montierten Werkzeugs auch einen Modus, in dem die programmierte Bahn durch extern überlagerte Daten verändert wird. Dieser Modus ist für den Fall erforderlich, daß im Arbeitsraum der Roboters Hindernisse auftauchen, oder sich die Lage der Objekte im Arbeitsraum ändert, um die Roboterbahn in Echtzeit¹ zu beeinflussen (Kollisionsvermeidung). Voraussetzung für die Echtzeit-Bahnbeeinflussung unter VAL-II sind geradlinige Roboterbewegungen oder DELAY²-Anweisungen. VAL-II ermöglicht grundsätzlich zwei Arten der Echtzeit-Bahnbeeinflussung, die Beeinflussung durch einen unter VAL-II ablaufenden Koprozeß und die Beeinflussung durch extern überlagerte Daten [UNIMATION]. Die Echtzeit-Bahnbeeinflussung über einen VAL-II-Koprozeß wird hier nicht weiter erläutert. Ausführlich beschrieben ist dieser Modus in der vorangegangenen Arbeit [DICKEN 92].

Die Bahnbeeinflussung wird in der Roboterprogrammiersprache VAL-II durch den Befehl

```
ALTER( Kanal, Modus, Subroutine, Priorität) Datenliste
```

ausgelöst.

¹ Der Begriff Echtzeit wird im Kapitel Echtzeitbetriebssysteme umfassend erläutert.

² DELAY ist eine Anweisung in der Programmiersprache VAL-II.

Die einzelnen Parameter haben folgende Bedeutung:

PARAMETER	Wert	Bedeutung
Kanal	0	Externer ALTER-Betrieb über Port 2
	-1	Interner ALTER-Betrieb (VAL II Koprozeß)
Modus	5-Bit-Wert	ALTER-Modus, Aufschlüsselung siehe unten.
Subroutine		Name einer Subroutine, die im Falle einer durch den externen Rechner signalisierten Ausnahmebedingung abgearbeitet wird. Ist kein Name angegeben wird das Programm im Falle einer Ausnahmebedingung abgebrochen.
Priorität		Prioritätslevel, indem die Reaktion auf eine Ausnahmebedingung abgearbeitet wird. Bedeutungslos, wenn keine Subroutine angegeben wird.
Datenliste		Optionale Datenliste, die mit der einleitenden VAL-Nachricht übertragen wird.

Der 5-Bit Wert des ALTER-Modus teilt sich folgendermaßen auf:

Modus		
Bit	Wert	Bedeutung
0	0	ALTER-Modus nichtkumulative Bahnänderungen
0	1	ALTER-Modus kumulative Bahnänderungen
1	0	ALTER-Eingaben in World-Koordinaten
1	0	ALTER-Eingaben in Tool-Koordinaten
2	0	VAL-II-Nachrichten enthalten keine Transformationsdaten
2	1	VAL-II-Nachrichten enthalten Transformationsdaten
3	0	Ausgabetransformation bezüglich aktuellem Tool-Tip_Setpoint
3	1	Ausgabetransformation bezüglich Tool-Tip-Location berechnet aus Gelenkwinkeln
4	0	ALTER-Eingaben sind mit Ausnahme des Controlbyte zu ignorieren
4	1	ALTER-Eingaben sind zur Bahnänderung zu verwenden

3.1 Bahnbeeinflussung durch extern überlagerte Daten

Die Robotersteuerung führt den Roboter über eine in VAL-II programmierte Bahn. Dem externen Steuerrechner wird durch den ALTER-Betrieb kontinuierlich die aktuelle Lage und Orientierung des Roboters übermittelt. Der externe Steuerrechner erfährt beispielsweise über eine Sensorik von der Lage eines Hindernisses¹. Aus diesen Daten berechnet der externe Steuerrechner die notwendige Bahnänderung, die in einem VAL-II-spezifischen Format an den VAL-II-Prozeß gesendet wird.

Der ALTER-Modus wird in zwei Varianten unterschieden:

- ◆ Kumulierender ALTER-Betrieb
Jeder Alterdatenwert bewirkt eine Bahnauslenkung *um* den gesendeten Wert. Ein Nullwert bewirkt keine Bahnänderung.

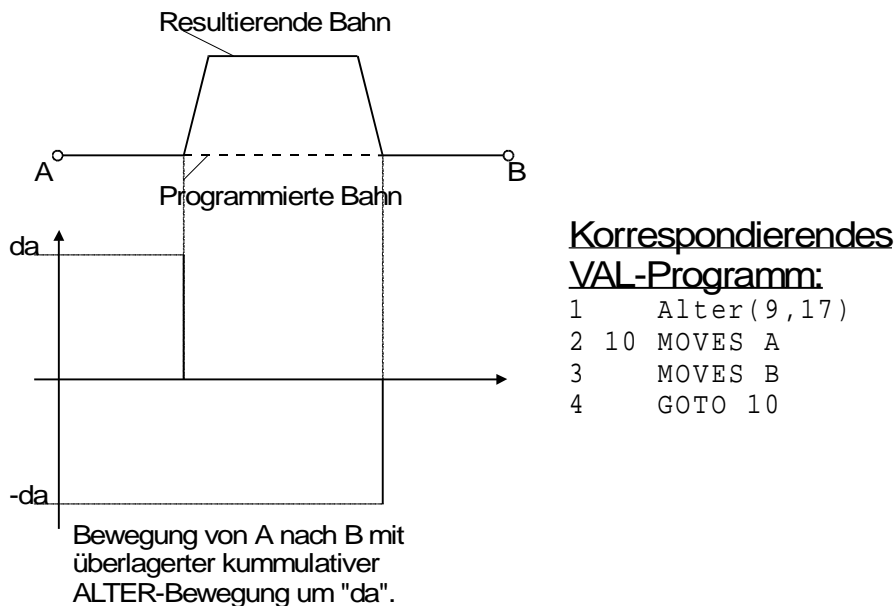


Bild 5. Kummulierender ALTER-Betrieb

¹ Der ALTER-Modus ist von UNIMATION ursprünglich für die Kollisionsvermeidung vorgesehen worden. Die hier beschriebene Implementation zweckentfremdet die ALTER-Schnittstelle für Teleoperation mittels einer Steuerkugel.

- ◆ Nichtkumulierender ALTER-Betrieb
 Jeder ALTER-Datenwert bewirkt eine Auslenkung *auf* den gesendeten Wert. Ein Nullwert bewirkt die Rückkehr in die Nullposition, d.h. auf die programmierte Bahn.

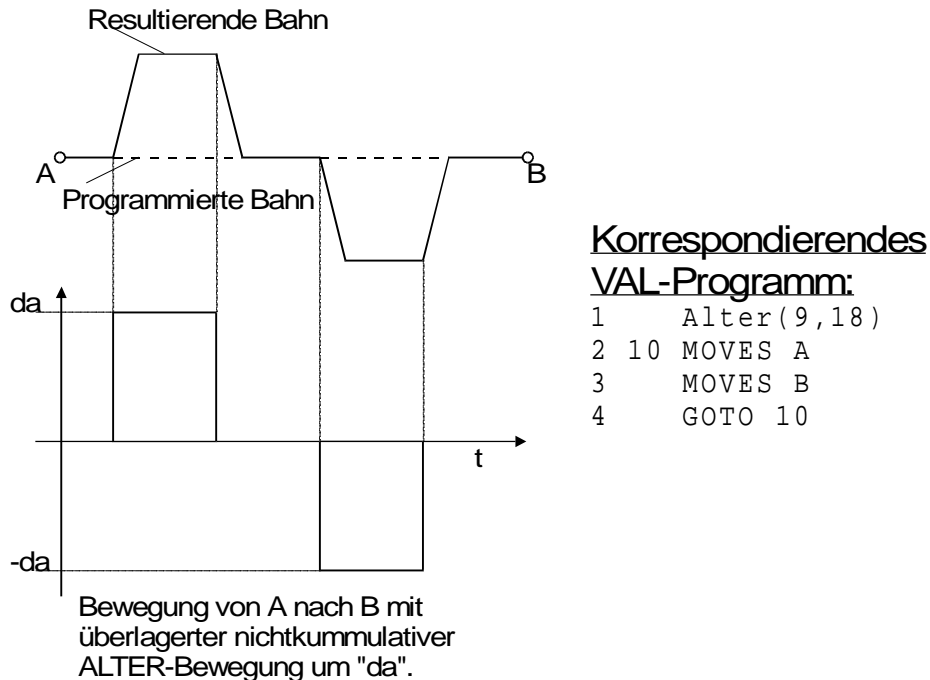


Bild 6. Nichtkumulierender ALTER-Betrieb

3.2 Kommunikationsprotokoll Robotersteuerung an externer Rechner

Die Kommunikation zwischen VAL-II und Steuerrechner erfolgt auf seriellen Wege über eine RS422-Schnittstelle mit der Baudrate 38400 im Datenformat 1 Startbit, 8-Bit-Zeichen, 1 Stopbit, keine Parität.

Nachdem die Robotersteuerung den VAL-II-ALTER-Befehl abgearbeitet hat, startet sie den Alterbetrieb. Im ALTER-Betrieb sendet die Robotersteuerung an den externen Rechner eine einleitende Mitteilung, die Kontrollinformationen zum verwendeten ALTER-Modus enthält. Der externe Rechner muß innerhalb einer bestimmten Zeitspanne¹ auf die einleitende Nachricht reagieren. Wenn keine Antwort des externen Rechners eintrifft, wird die einleitende Nachricht noch bis zu zweimal wiederholt. Scheitert auch die dritte Kommunikationsaufforderung wird der ALTER-Betrieb mit einer Fehlermeldung beendet. Wenn allerdings der externe Rechner im festgesetzten Zeitrahmen auf die einleitende Nachricht antwortet, fährt die Robotersteuerung im ALTER-Modus fort und sendet in regelmäßigen Abständen Nachrichten an den externen Rechner, die nun Kontrollinformationen über den aktuellen Bewegungsablauf und die aktuelle Position enthalten. Jede dieser Nachrichten muß wiederum in einer bestimmten Zeit vom externen Rechner beantwortet werden, um den ALTER-Modus aufrecht zu erhalten.

¹ Der bei der Kommunikation einzuhaltende Zeitrahmen wird im Kapitel 3.4 Zeitverhalten erläutert.

Zusammen mit diesen Bestätigungen übersendet der externe Rechner die aktuellen ALTER-Daten. Diese Kommunikation wiederholt sich solange, bis entweder der externe Rechner oder die Robotersteuerung durch eine Unterbrechungsmitteilung den ALTER-Betrieb beenden.

Die Nachrichten zum und vom externen Rechner werden in Form von Datenpaketen ausgetauscht, die in folgender Weise gekapselt sind:

DLE DLE STX <Daten> DLE ETX <CRC>

<CRC> stellt eine einfache Prüfsumme Modulo 256 über die Daten des Paketes dar. Die Zeichen DLE, STX und ETX sind Bestandteil des ASCII-Zeichensatzes und haben die Bedeutung:

Name	Hexadezimal	Dezimal	Bedeutung
DLE	010h	16	Data Link Escape
STX	02h	2	Start of text
ETX	03h	3	End of text

Die Kapselung erhöht die Zuverlässigkeit des Protokolls und ermöglicht eine einfache Wiederaufnahme der Kommunikation bei Empfang eines defekten Pakets. Da ein DLE-Zeichen auch innerhalb des Datenfeldes vorkommen kann, wird im Datenfeld jedem vorkommenden DLE ein zweites hinzugefügt. Ein Auftreten von STX und ETX Zeichen im Datenfeld ist unkritisch, weil diese Zeichen in Kombination mit dem DLE Zeichen auftreten müssen um für das Paket-Protokoll relevant zu sein.

3.2.1 Einleitende Nachricht an den externen Rechner

Das Datenpaket der einleitenden Nachricht an den externen Rechner hat folgende Form:

Byte	Bezeichnung	Beschreibung
1	Control-Byte	Kontrollinformationen an den externen Rechner Bit 0 bis 2 Error-Code Bit 5 bis 6 Alterstatus
2	Alter-Modus	Parameter für Alter-Charakteristik (entspricht dem Wert Alter-Modus-Wert aus dem Alter-Befehl, Beschreibung siehe Tabelle x-x)
3 bis n	Daten	Integer-Werte aus der optionalen Datenliste der Alter-Instruktion, je zwei Byte pro Integer-Wert, erst Low-Byte dann High-Byte.

3.2.2 Standardnachricht an den externen Rechner

Auf die einleitende Nachricht folgen nach Bestätigung durch den externen Steuerrechner Standardnachrichten dieser Struktur:

Byte	Bezeichnung	Beschreibung
1	Control-Byte	<p>Kontrollinformationen an den externen Rechner</p> <p>Bit-Format: 1mm00eee mm: Art der Mitteilung 00: ALTER läuft 01: ALTER beginnt 10: ALTER unterbrochen 11: ALTER beendet</p> <p>eee: Fehlercode 000: Kein Fehler 001: Checksum-Fehler 010: Formatfehler 011: Überlauf Datenpuffer 100: Letzte Mitteilung unvollständig 101: Protokollfehler in Anfangs- oder Endsignatur 110: Timeout-Fehler</p>
2	Segment-Status	Status des aktuellen Bewegungsschrittes
		Wert Interpretation
		1 Es wurden noch keine Bewegungsanweisungen ausgeführt
		2 Normale Bahnbestimmung ist im Ablauf
		3 Bewegung ist beendet, Location erreicht
		4 Positionsfehler wird ungültig, es wurde eine nicht vorgesehene Location erreicht
		5 Bewegung gestoppt an nicht vorgesehener Location
		6 Bremsend bis HALT-Modus erreicht. Gestoppt im HALT-Modus
		7 Bremsend wegen REACTI- oder BREAK-Anweisung
		8 Stoppend wegen REACTI- oder BREAK-Anweisung
		9 Bremsend wegen Hardwarefehler oder "Panic Stop"
		10 Stoppend wegen Hardwarefehler oder "Panic Stop"
		11 Bremsend als Reaktion auf Signal
12 Stoppend als Reaktion auf Signal		
13 Roboterarm befindet sich nahe der programmierten Position		

3 bis 4	Segment-Nummer	Wird um 1 für jede neue Bewegungsanweisung im internen VAL-Programm erhöht	Beginnt mit 1 und wird um 1 erhöht, wenn der Übergang von einer Bewegungsanweisung zur nächsten stattfindet
5 bis 6	Bewegungs-Prozentsatz	Prozentsatz, mit dem das aktuelle Segment bereits ausgeführt worden ist	(100% -> 40000 Oktal) der Übergang zum nächsten Segment kann bereits dann stattfinden, wenn der Wert noch ein wenig unter 100% liegt. Dieses ist von der Bahn und der Robotergeschwindigkeit abhängig.
7 bis 30	Transformationsbytes	Transformationsdaten	Entsprechend dem Modus-Parameter bei der Ausführung der ALTER-Anweisung

3.2.3 Abschließende Nachricht der Robotersteuerung an den externen Rechner

Die Robotersteuerung überträgt zum Verbindungsabbau ein Control-Byte mit dem Wert "ALTER-Stopping". Der externe Rechner muß darauf nicht antworten.

3.3 Kommunikationsprotokoll externer Rechner an Robotersteuerung

Der externe Steuerrechner antwortet auf die verschiedenen Kommunikationsanforderungen der Robotersteuerung in folgender Weise.

3.3.1 Antwort des externen Rechners auf einleitende Nachricht

Die Antwort an die Robotersteuerung besteht nur aus einem einzigen Control-Byte:

Wert	Bedeutung
1	Der externe Rechner hat einen Kommunikationsfehler entdeckt und erwartet, daß die Robotersteuerung die einleitende Nachricht wiederholt. Nach der dritten Wiederholung stoppt VAL-II mit einer Fehlermeldung.
0	Der externe Rechner hat die einleitende Nachricht korrekt empfangen und ist bereit zur Aufnahme des normalen ALTER-Betriebes.
-1	Zeigt an, daß der externe Rechner den ALTER-Modus nicht aufnehmen möchte. VAL-II beendet das Programm sofort mit einer Fehlermeldung.

3.3.2 Antwort des externen Rechners auf Standardnachricht

Byte	Name	Beschreibung
1	Exception Code	Anwendungsspezifischer Code, der das mit der ALTER-Anweisung spezifizierte Unterprogramm aktivieren kann. Dieses Byte sollte während des normalen ALTER-Betriebes immer auf Null gesetzt werden.
2	Select Bit	Jedes gesetzte Bit zeigt an, daß der zugehörige ALTER-Wert mit dieser Nachricht übertragen wird. Im kumulativen Betrieb wird jedes nicht übertragene Datenwort von VAL-II intern unverändert gelassen. Anfangs werden alle Werte als Null angenommen.
3 bis 14	ALTER-Werte	Jedes 16-Bit-Wort gehört zu einem gesetzten Bit in Select-Bit-Byte. Die Worte werden als je zwei Byte Übertragen, beginnend mit dem Low-Byte. Die Werte werden in folgender Reihenfolge übertragen: 1 bis 3 Translation X, Y, Z 4 bis 6 Rotation X, Y, Z

Die durch die ALTER-Werte ausgelösten Bewegungen ergeben sich durch die Division mit einem roboterspezifischen Skalierungsfaktor.

ALTER-Bewegung	Skalierungsfaktor	Erzielbarer Wertebereich	Beispiel
Translation	32	+/- 1024 mm	ALTER-Wert T; Bewegung Roboter B $B[\text{mm}] = T/32.0[\text{mm}]$
Rotation	182.044	+/- 180 °	ALTER-Wert R; Bewegung Roboter D $D[^\circ] = R/182.0444[^\circ]$

3.3.3 Abschließende Nachricht vom externen Rechner

Eine abschließende Nachricht vom externen Rechner ist nicht erforderlich, da die Robotersteuerung bei Ausbleiben von Antwortpaketen den ALTER-Betrieb beendet.

3.4 Zeitverhalten

Auf die Nachrichten der Robotersteuerung muß der externe Steuerrechner innerhalb eines bestimmten Zeitrahmens reagieren, damit der ALTER-Modus aufrechterhalten wird. Die Robotersteuerung erwartet, daß während des ALTER-Betriebes alle 28 ms ein Kommunikationshandshake durchgeführt wird. Wird die Übertragungsdauer der anfordernden Nachrichtenpakete berücksichtigt, verbleiben dem externen Rechner noch 15 ms im kumulativen bzw. 17 ms im nichtkumulativen Betrieb, in denen die Antwortnachricht komplett übertragen worden sein muß.

3.4.1 Übertragungsdauer der Nachrichten-Pakete an den externen Steuerrechner

Die Übertragungsdauer der ALTER-Anforderungspakete an den externen Steuerrechner ist im wesentlichen davon abhängig, ob die Transformationsdatenübertragung in der ALTER-Anweisung gewählt wurde. Für eine Baudrate von 38400 und maximal mögliches Byte-Stuffing ergeben sich folgende längstmögliche Übertragungszeiten:

Transformationsdaten	Länge (Bytes)	Zeit (ms)
keine	14	3,6
mit	52	13,5

3.4.2 Übertragungsdauer der Nachrichten-Pakete des externen Steuerrechners

Die Zeitspanne, die der externe Rechner zur Verarbeitung der Nachricht benötigen darf, hängt von der Anzahl der zu übertragenden ALTER-Werte zur Bahnbeeinflussung ab. In folgender Tabelle sind die maximalen Zeitspannen (inklusive Byte-Stuffing) vor Übertragungsbeginn der Antwortnachricht an die Robotersteuerung aufgelistet [UNIMATION].

Anzahl der gesetzten Selectbits	Kumulativer ALTER-Betrieb		Nichtkumulativer ALTER-Betrieb	
	Bytes	Zeit	Bytes	Zeit
0	8	10,8	8	12,8
1	11	9,2	12	10,7
2	14	7,7	16	8,6
3	17	6,1	20	6,5
4	20	4,5	24	4,5
5	23	3,0	28	2,4
6	26	1,4	32	0,3

4 Kommunikation mit der Steuerkugel

Gegenüber der ursprünglichen Implementierung wurde der Gesamtaufbau durch eine weitere 6-Dimensionen-Steuerkugel ergänzt, um eine komfortable Veränderung der Szenenansicht im Visualisierungsprogramm SMS zu ermöglichen. Bei der neuen Steuerkugel vom Typ ISRA-Robot-Teach-Ball handelt es sich um ein anderes Fabrikat mit veränderten Kommunikationseigenschaften. Da SMS die vorhandene Steuerkugel vom Typ DIMENSION 6 als Eingabegerät bereits unterstützt, wurde sie das Kommunikationsprotokoll der neuen Steuerkugel in das Leitrechnerprogramm VAL5 integriert. Ebenso, wie die DIMENSION 6, verfügt der ISRA-Robot-Teach-Ball über eine serielle RS-232 Schnittstelle zur Kommunikation mit dem Leitrechner. Die VAL5-Software ermöglicht es nun, Steuerkugeln beider Typen wechselweise zu verwenden, wobei der Steuerkugeltyp dann im Menü Optionen eingestellt werden kann.

Ebenso wie wir die Steuerkugel vom Typ DIMENSION 6, ermöglicht der ISRA-Robot-Teach-Ball über die Betätigung der Tasten "ROT", "TRA" und "DOM", die Einstellung der bevorzugt zu übertragenden Bewegungsrichtungen.

Taste	Funktion
ROT	Nur der rotatorische Anteil der angreifenden Kräfte/Spannungen wird berücksichtigt
TRA	Nur der translatorische Anteil der angreifenden Kräfte/Spannungen wird berücksichtigt
DOM	Nur die jeweils dominierende Komponente der angreifenden Kräfte/Spannungen wird berücksichtigt. Bei gleichzeitiger Aktivierung der ROT- oder TRA-Taste wird nur die dominierende Komponente des rotatorischen- oder translatorischen Anteil der Kräfte/Spannungen berücksichtigt

4.1 Hardwareeinstellungen für den ISRA-Robot-Teach-Ball

An der Unterseite des ISRA-Robot-Teach-Balls befinden sich acht DIP-Schalter für die Einstellung der Kommunikationsparameter. Für die Implementation wurden folgende Parameter ausgewählt:

- ◆ 9600 Baud Übertragungsrate
- ◆ Datenanforderung mit einem Byte
- ◆ "TRA", "DOM", "ROT"-Tastenwerte werden nicht übertragen
- ◆ Kraft-/Momentenwerte werden übertragen

Daraus ergibt sich folgende Einstellung:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-------	-------	-------	-------	-------	-------	-------	-------

ON	ON	X	X	X	OFF	ON	OFF
----	----	---	---	---	-----	----	-----

X : don't care

Die serielle Schnittstelle des ISRA-Robot-Teach-Balls arbeitet im Halbduplexbetrieb mit integriertem XON/XOFF Software-Handshake.

4.2 Kommunikationsprotokoll ISRA-Robot-Teach-Ball

Im Gegensatz zur DIMENSION 6 Steuerkugel verfügt der ISRA-Robot-Teach-Ball über kein Kommunikationsprotokoll, welches Daten *ohne* Aufforderung an den Steuerungsrechner sendet. Der ISRA stellt zwei verschiedene Kommunikationsprotokolle zur Verfügung: [HOF88]

- ♦ der Steuerrechner fordert Daten vom ISRA-Ball mit einem Daten-Telegramm an
- ♦ der Steuerrechner sendet zur Anforderung ein einzelnes Enquire-Byte (ENQ)

Für die Implementation wurde die Anforderung von Daten mittels eines einzelnen Enquire-Bytes gewählt um die Kommunikationsbelastung des Steuerrechners möglichst niedrig zu halten. Der ISRA-Robot-Teach-Ball sendet nach Empfang eine ENQ-Bytes in diesem Modus ein Antwortpaket, das folgendermaßen aufgebaut ist (jede Zeile ein Byte):

n Anzahl der Datenbytes ohne Checksumme
-n
n XOR 55h
ID-Byte
X-Kraft
Y-Kraft
Z-Kraft
X-Moment
Y-Moment
Z-Moment
Anwendertasten
Checksumme

Das ID-Byte ist so kodiert :

ab1	Sensor-Typ Bit 2	Sensor-Typ Bit 1	Sensor-Typ Bit 0	ab0	Sensor-Nr.Bit 2	Sensor-Nr.Bit 1	Sensor-Nr.Bit 0
-----	------------------	------------------	------------------	-----	-----------------	-----------------	-----------------

Wobei "ab1" und "ab0" in diesem Modus folgende Zustände annehmen können:

ab1	ab0	Rückmeldungstyp
0	0	(Anforderung / Daten) in Ordnung
1	0	alte Daten

Bedeutung der Rückmeldungen:

Daten in Ordnung : Die Datenanforderung war syntaktisch in Ordnung. Es folgt die Übertragung eines Datenpaketes.

Alte Daten : Korrekte Datenanforderung, aber der Prozessor des Teach-Balls hat die Generierung neuer Daten noch nicht abgeschlossen. Es werden die vorhergehenden Daten übermittelt.

Die Bytewerte der Kräfte/Momente lassen sich als 8 Bit Integerzahlen in Zweier-Komplement-Darstellung im Bereich von -128.....+127 interpretieren.

Kodierung der Anwendertasten:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Taste 8	Taste 7	Taste 6	Taste 5	Taste 4	Taste 3	Taste 2	Taste 1

Taste: 0 -> nicht gedrückt
1 -> gedrückt

Berechnung der Checksumme:

$$(n + \text{Summe der Datenbytes}) \text{ MOD } 256 + \text{Checksumme} = 0$$

5 Kommunikation des Steuerrechners mit dem Visualisierungssystem

Für die Visualisierung der Teleoperation steht die beschriebene Apollo-Workstation zur Verfügung. Die Visualisierungssoftware SMS liegt in einer Version mit spezieller Softwareschnittstelle vor, die es erlaubt, Kommunikationsroutinen zu Implementieren, um eine Fernsteuerung des simulierten Roboters vom externen Steuerrechner zu ermöglichen. Die ursprüngliche Implementierung dieser Kommunikationsroutinen sieht folgendes einfache Protokoll vor [DICKEN92]:

Die Workstation sendet nach erfolgreichem Laden und Starten der PUMA-562-Simulationsumgebung unter SMS ein ENQ (Enquire)- Byte, das den externen Kommunikationsrechner auffordert, aktuelle Transformationsdaten zu senden. Die Transformationsdaten setzen sich aus den aktuellen translatorischen X-, Y-, und Z-Koordinaten und der entsprechenden rotatorischen Orientierung des Greifers zusammen. Die Workstation führt eine Rücktransformation dieser Koordinaten in Roboterachsenpositionen durch und stellt die neue Position des Roboters in der Simulation dar. Die Neuberechnung der Simulation in schattierter (gerenderter¹) Darstellung erfordert auch mit der verwendeten Hochleistungsgrafik eine Rechenzeit von ca. 0.2 Sekunden, so daß der Visualisierungsrechner nach dieser Zeitspanne sofort wieder ein neues ENQ-Byte an den externen Steuerrechner sendet.

Die Rücktransformation und die Kommunikationsroutinen unter SMS wurden bereits in der Arbeit zur ursprünglichen Implementation ausführlich beschrieben. (Siehe [DICKEN92])

Auf physikalischer Ebene wird die Verbindung zwischen Grafik-Workstation und externem Steuerrechner mittels RS232-Schnittstellen realisiert. Die Kommunikation erfolgt mit der Baudrate 19200 im Datenformat 1 Startbit, 8-Bit-Zeichen, 1 Stopbit, keine Parität. Als Handshaking-Verfahren wird Software-Handshaking mit XON/XOFF-Zeichen verwendet. Auf Workstationseite wurde die Schnittstelle SIO3 verwendet.

Die Verwendung der Softwareschnittstelle von SMS ist über die Routinen "ball.pas" und "puma_koll_verm.c" realisiert worden. Diese beiden Routinen liegen im Quelltext vor und können nach Compilation als Object-Dateien an einen, um diese Funktionen reduzierten, SMS-Kern gebunden werden. Die PASCAL-Funktion "ball.pas" ist ursprünglich für die Kommunikation von SMS mit einer Steuerkugel vom Typ DIMENSION 6 vorgesehen und übernimmt Initialisierungen der seriellen Schnittstelle. Hier wurde die Initialisierung der Kommunikation für die neu verwendete serielle Schnittstelle eingebaut. Die C-Funktion "puma_koll_verm.c" dient ursprünglich zur Kollisionsvermeidung in der SMS-Simulationsumgebung und wird, eine eingeschaltete Kollisionsüberprüfung und ein minimales ablaufendes Robotersimulationsprogramm vorausgesetzt, in jedem

¹ Die Simulationsumgebung erlaubt die Definition von Lichtquellen, um mit Hilfe von Beleuchtungsalgorithmen eine realistisch gerenderte 3-D-Darstellung zu ermöglichen. Die Berechnung des Beleuchtungszustandes ist bei beweglichen Objekten für jedes Einzelbild neu erforderlich. Die in solchen Beleuchtungsberechnungen verwendeten Algorithmen stellen nicht unerhebliche Anforderungen an die Rechenleistung eines solchen Systems. Erst moderne Silicon-Graphics-Workstations erlauben eine derartige Darstellung in Echtzeit. (25 Bilder pro Sekunde)

Darstellungszyklus einmal ausgeführt. In diese Funktion wurde das Kommunikationsprotokoll integriert.

Das auf der Apollo-Workstation im SMS-Simulationsbetrieb ablaufende PUMA-Programm hat folgenden einfachen Aufbau:

```
WAIT SIG(1001)
```

Die genauen Modifikationen dieser Routinen sind in der Arbeit [DICKEN92] ausführlich beschrieben.

6 Echtzeitdatenverarbeitung

6.1 Begriffserläuterungen

Hier sollen zunächst die Begriffe "Multitasking", "Time-Sharing" und Echtzeitbetriebssystem kurz erläutert werden.

6.1.1 Multitasking

Der Begriff Task bezeichnet eine Aufgabe oder einen Prozeß, der vollständig sequentiell abgearbeitet wird. Multitasking bedeutet, daß mehrere solcher Tasks parallel abgearbeitet werden. Ein-Prozessor-Systeme sind nicht in der Lage, mehrere Tasks gleichzeitig ablaufen zu lassen, sondern erfordern eine Umschaltung zwischen den einzelnen Tasks, um so eine quasi parallele Verarbeitung zu ermöglichen. Häufig ist eine Kooperation der einzelnen Tasks erforderlich, so kann es notwendig werden, daß eine Task erst dann weiterlaufen kann, wenn eine andere einen bestimmten Arbeitsschritt erreicht hat. In einem solchen Fall ist eine Synchronisation mit Hilfe von Inter-Task-Kommunikation notwendig, die Parallelität wird wieder eingeschränkt.

Man unterscheidet zwischen kooperativen- und preemptiven Multitasking.

Kooperatives Multitasking ist ein Verfahren, bei dem Tasks nur unterbrochen werden können, wenn sie dazu bereit sind. "Unhöfliche" oder fehlerhafte Tasks können daher das gesamte System anhalten. Deshalb ist dieses Verfahren nicht für einen Echtzeitbetrieb geeignet.

Preemptives Multitasking ist ein Verfahren, bei dem Tasks jederzeit unterbrochen werden können. Dieses Verfahren ist für den Rechner etwas aufwendiger, aber insgesamt leistungsfähiger und zuverlässiger.

6.1.2 Time-Sharing

Time-Sharing ist der Einsatz von Multitasking um auf einem Rechner mehreren Benutzern (oder Batch-Jobs) gleichzeitig Rechenzeit zur Verfügung zu stellen. Die Tasks laufen relativ unabhängig voneinander ab, Inter-Task-Kommunikation wird nur in einfacher, nicht echtzeitfähiger Form geboten. Beispiele für Time-Sharing-Systeme sind UNIX-Systeme auf denen Rechenzeit zwischen Benutzern "gerecht" verteilt wird. Time-Sharing-Systeme garantieren auch bei Überlastung (die zur Verfügung stehende Rechenzeit wird voll ausgeschöpft) eine "gerechte" Verteilung der Rechenzeit auf die verschiedenen Benutzer, die Reaktionszeiten für jeden einzelnen Benutzer wird allerdings größer.

6.1.3 Echtzeit-Betriebssysteme

Echtzeit- ("Real-Time") Verarbeitung bedeutet nicht, wie oft angenommen wird, unbedingt eine hohe Geschwindigkeit, sondern eine in jedem Fall garantierte Antwortzeit. Das System muß seine Aufgaben zeitgerecht innerhalb einer vorgegebenen maximalen Zeit erledigen. Insbesondere muß die Reaktion auf externer Unterbrechungsanforderungen (Interrupts) diesen Rahmenbedingungen genügen. Der ungünstigste Fall ("Worst-Case") der Unterbrechungsanforderungen muß analysiert werden um sicherzustellen, daß das System den Anforderungen genügt. Ein Echtzeitbetriebssystem muß also nicht unbedingt sehr schnell, sondern nur schnell genug für die vorgesehene Anwendung sein. Auch Systemdienste, wie z.B. Festplatten oder Netzwerkzugriffe, müssen unterbrechbar sein.

Echtzeitbetriebssysteme unterscheiden sich grundsätzlich von Time-Sharing-Systemen. Solche Betriebssysteme dürfen niemals überlastet sein, denn durch Überlastung verliert ein Echtzeitbetriebssystem seine Echtzeitfähigkeit. Echtzeitsysteme verteilen die Rechenzeit nicht nach "Gerechtigkeit", sondern einzelne Tasks haben Prioritäten, welche respektiert werden müssen. Eine Task mit hoher Priorität kann jeder Zeit einer niedrigpriorisierten Task Rechenzeit entziehen. Weil keine Überlastung existiert, kommt auch die niedrigpriorisierte Task zu ihrer CPU-Zeit.

Zusammenfassend werden an ein Echtzeitbetriebssystem folgende Anforderungen gestellt:

- ◆ Garantierte maximale Reaktionszeit auf äußere Ereignisse
- ◆ Beendigung von Aufgaben innerhalb einer bestimmten Zeit
- ◆ Verwaltung unterschiedlicher Prioritäten von Ereignissen und Aufgaben
- ◆ Hohes Maß an Zuverlässigkeit
- ◆ Geringe Interrupt-Latenzzeit (Zeit, in der keine Interrupts bearbeitet werden können)

6.2 Serielle Echtzeitkommunikation unter MSDOS

Das Betriebssystem MSDOS stellt von sich aus nur Betriebssystemaufrufe in Form von Portabfragen zur Verfügung. Um Echtzeitbedingungen zu erfüllen, müssen auch in Hochsprachen Interrupt-Service-Routinen (ISR) geschrieben werden. Dieser Ansatz wurde in der ursprünglichen Portierung verfolgt. Die Kodierung einer ISR erfordert hardwarenahe Programmierung von Portbausteinen und Interruptcontrollern. Die MSDOS Betriebssystem-Erweiterungen, bzw. grafischen Benutzeroberflächen WINDOWS 3.x, oder WINDOWS 95 bieten ebenfalls keine garantierten Antwortzeiten auf Systemereignisse.

6.3 Das Echtzeitbetriebssystem OS-9

OS-9 ist ein UNIX-ähnliches Multiuser-, Multitasking-, Echtzeitbetriebssystem, das von der Firma MICROSYS ursprünglich für den 8-Bit MC-6809 Prozessor entwickelt worden ist. Später wurde OS-9 auf die MOTOROLA 680x0 Prozessoren portiert und kontinuierlich weiterentwickelt. OS-9 ist vollständig in Maschinensprache geschrieben¹, wodurch eine hohe Arbeitsgeschwindigkeit erreicht wird. Das Betriebssystem selbst und alle Programme für OS-9 sind in sogenannten Modulen organisiert. Diese Module werden mit Hilfe des OS-9-Dienstprogrammes "MODED" (Moduleditor) vom Linker oder durch Hochsprachen erzeugt. Alle Module haben eine gemeinsame Struktur und die Eigenschaft relozierbar (frei im Speicher verschiebbar) zu sein. Außerdem dürfen sie keinen sich selbst veränderlichen Programmcode enthalten. Dies sind die Voraussetzungen für die effektive Speicherverwaltung von OS-9.

Beim Startvorgang überprüft das Betriebssystem den verfügbaren Speicher und unterscheidet zwischen verschiedenen RAM (Random-Access-Memory)-Bereichen (batteriegepuffert, Video, System), sowie ROM (Read-Only-Memory)-Bereichen. Gleichzeitig sucht es die verschiedenen Speicherbereiche nach gültigen Modulen ab, welche mittels eines speziellen Synchronisationsworts (4AFCh) von anderen Speicherbereichen zu unterscheiden sind. Wenn OS-9 beim Initialisieren des Speichers auf dieses Wort trifft, versucht es ab dieser Stelle im Speicher einen 30h -Byte langen Modulkopf zu lesen, der durch eine Prüfsumme abgeschlossen ist. Ist diese Prüfsumme korrekt, gibt ein Längeneintrag im Modulkopf an, über den ein CRC-Wert² zu bestimmen ist. Nur wenn dieser mit den letzten vier Bytes des Moduls übereinstimmt, wird das Modul als gültig akzeptiert und in das Modulverzeichnis eingetragen. Auf diese Weise erkennt OS-9 auch Module, welche im ROM eines Systems gespeichert sind. OS-9 und Anwendungsprogramme unter OS-9 lassen sich vollständig oder teilweise im ROM betreiben, ohne daß sich die Notwendigkeit eines Massenspeichers ergibt. Daraus folgt die Eignung von OS-9 für Embedded-Controller-Anwendungen, Beispielsweise laufen die Controller der meisten CD-I-Player (Compact-Disk-Interactive) unter dem OS-9-Betriebssystem.

¹ Im Gegensatz dazu sind beispielsweise gängige UNIX-Dialekte zum aller größten Teil in der Hochsprache C geschrieben (über 80 %), um eine rasche Portierung auf verschiedene Hardwareplattformen zu ermöglichen. Auch MICROWARE geht diesen Weg bei der Neuentwicklung OS-9000, welche komplett in C neu geschrieben worden ist, und so für sehr unterschiedliche Prozessoren zur Verfügung steht.

² .CRC ist die Abkürzung für cyclic redundancy check, einem sehr zuverlässigen Testverfahren auf Datenintegrität.

Das Betriebssystem OS-9 untergliedert sich in drei Teile:

- ◆ Kern,
- ◆ Filemanager,
- ◆ Treiber und Deskriptoren.

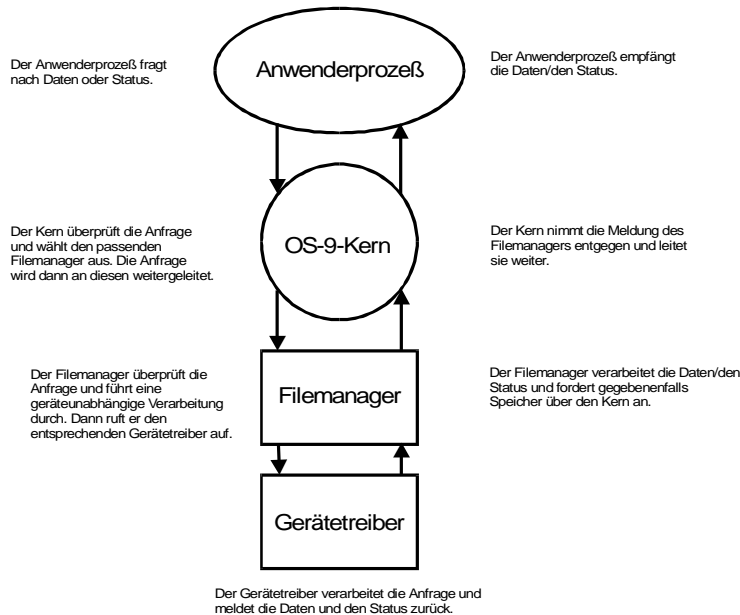


Bild 7. OS-9 Struktur (Quelle: CT1/95)

Der Betriebssystemkern von OS-9, der lediglich 26 KByte umfaßt, enthält den Scheduler, die wichtigste Funktionseinheit eines Echtzeit-Multitaskingbetriebssystems. Der Scheduler entscheidet, welche Task als nächste laufen darf und schaltet zwischen den Tasks um. Außerdem sorgt der Kern für alle Basisfunktionen des Betriebssystems, wie beispielsweise das Management der Ein- und Ausgaben, die Verwaltung der Betriebsmittel und Devices, sowie die Kontrolle über die laufenden Programme und ihre Benutzer. OS-9 läßt sich leicht auf eine neue Hardwareumgebung portieren, weil die Anpassung des Kerns an die Hardware durch Initialisierungsmodule erfolgt. Um den Kern herum gruppieren sich die sogenannten Filemanager und Traphandler. Die Filemanager verarbeiten Daten, die der Kern und die jeweiligen Gerätetreiber für eine Klasse von ähnlichen Geräten austauschen. So erkennt beispielsweise der Random-Block-Filemanager (RBF) als einziger Bestandteil von OS-9 die Struktur einer Festplatte. Er verwaltet die gerätespezifische Struktur und das Dateisystem auf dem Speichermedium. Weiterhin gibt es Filemanager für zeichenorientierte Geräte (SCF für Drucker, Terminal, Modem) oder für die Kommunikation zwischen Prozessen. Jeder Filemanager hat genau definierte Programmierschnittstellen, die das Erstellen eines neuen Treibers sehr einfach gestalten. Ein Gerätetreiber ist der Teil des Betriebssystems, der einen direkten Bezug zur angeschlossenen Hardware hat, z.B. werden Register von I/O-Portbausteinen hier definiert. Jeder Treiber enthält Routinen zur Initialisierung, Eingabe, Ausgabe, Terminierung und Interrupt-Service. Die Gerätetreiber bedienen sich sogenannter Gerätedeskriptoren, welche die genauen Daten der angeschlossenen Geräte enthalten. [CT1/95]

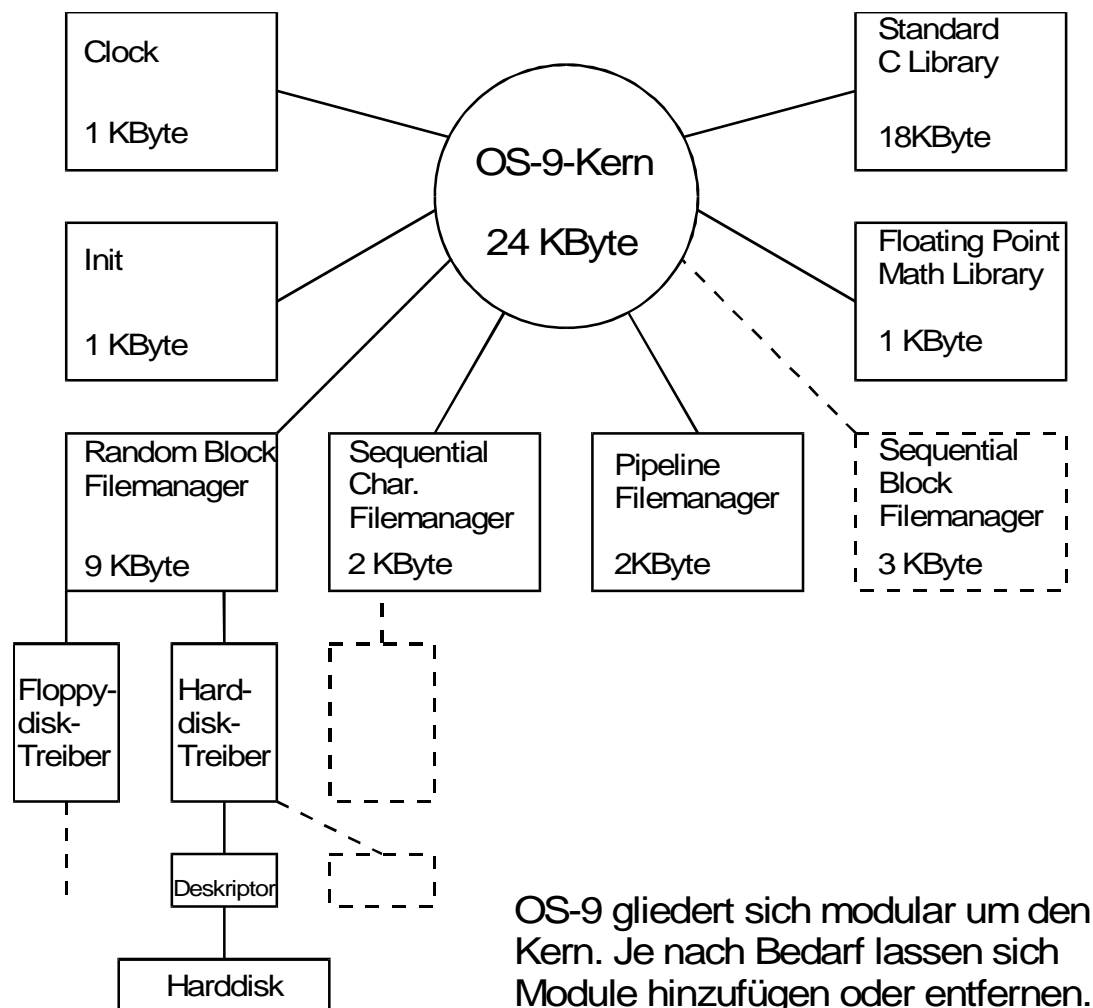


Bild 8. Modularer Aufbau von OS-9 und Speicherbedarf (Quelle CT1/95)

6.4 Preemptives Multitasking unter OS-9

Nachdem Programme von OS-9 in den Speicher geladen und in das Modulverzeichnis eingetragen worden sind, sind sie als Prozeß lauffähig. Ein Prozeß befindet sich immer in einem der Zustände aktiv, wartend oder schlafend. Für jeden dieser Zustände existiert eine Liste, welche die Deskriptoren der entsprechenden Prozesse enthält. Die Zustandsänderung eines Prozesses erfolgt durch den Transfer des Prozeßdeskriptors von einer Liste in die andere. Die Aktiv-Liste führt alle Prozesse auf, die auf eine Zeitscheibe warten. Wird ein Prozeß durch den OS-9-Funktionsaufruf (`F$WAIT`) in den Wartezustand versetzt, verharrt er solange, bis er durch ein 'Wake-Up'-Signal oder die Beendigung eines Tochterprozesses wieder aktiviert wird. Der Funktionsaufruf `F$SLEEP` versetzt einen Prozeß dagegen in den Schlafzustand, der eine bestimmte Anzahl von Zeitscheiben andauert. Nach Ablauf dieser Frist, deren Länge als

Parameter angegeben wird, wechselt der Prozeß automatisch wieder in die Aktiv-Liste. Diese Mechanismen finden immer dann Anwendung, wenn ein Prozeß auf Ein- oder Ausgaben wartet, die nicht sofort stattfinden können. Ruft ein Prozeß beispielsweise die C-Funktion `getchar()` auf und es befindet sich kein Zeichen im Tastaturpuffer, so schaltet OS-9 ihn auf "wait" und widmet sich nur noch den übrigen, aktiven Prozessen. In dem Augenblick, indem die Tastatur betätigt wird, stellt das Betriebssystem fest, daß ein Prozeß eben darauf wartet und weckt ihn auf, so daß er bei der nächsten Zeitscheibenvergabe wieder aktiviert wird. Der Wait-Sleep-Mechanismus hat den Vorteil, daß OS-9 keine Zeit mit sinnlosen Wartevorgängen (Polling) verbringt, sondern die gesamte CPU-Leistung den aktiven Prozessen zur Verfügung stellt.

6.5 Prozeßprioritäten unter OS-9

Durch Vergabe von Prioritäten wird erreicht, daß Prozesse mit einer hohen Priorität häufiger eine Zeitscheibe erhalten, als solche mit einer niedrigen. Die standardmäßig verwendeten Zeitscheiben mit einer Länge von 10 Millisekunden können vom Anwender an spezielle Bedürfnisse angepaßt werden. Grundlage für die Verteilung der Zeitscheibe durch den Scheduler ist das Prozeßalter, welches sich aus der Priorität und der Zeitdauer, die der Prozeß bereits auf seine Ausführung wartet, zusammensetzt. Das Auftreten von Events (Ereignissen) und Interrupts bewirkt die neue Verteilung von Rechenzeit, um vorgegebene Antwortzeiten garantieren zu können. Aktiviert ein Interrupt oder ein Event eine Task, welche eine höhere Priorität als die gerade laufende hat, so verliert diese ihre noch verbleibende Rechenzeit und wird neu in die Scheduler-Liste eingeordnet. Neben Interrupts beeinflussen auch die globalen Variablen `D_MinPty` und `D_MaxPty` die Ausführung der einzelnen Tasks. Prozesse deren Priorität niedriger ist als `D_MinPty` werden nicht mehr ausgeführt, solange Prozesse mit Prioritäten über diesem Wert existieren. `D_MaxPty` legt ein Alter fest, über das hinaus ein Prozeß nicht altern kann. Läuft ein Prozeß mit einer höheren Priorität, so ruht während dessen Ausführung das Time-Sharing. [CT1/95]

6.6 Gegenüberstellung MS-DOS - OS-9

Zweifellos kann man unter MS-DOS mittels hardwarenaher Programmierung echtzeitfähige ISR programmieren. Im Vergleich zu den eleganten Möglichkeiten die OS-9 in den Bereichen Prozeßsynchronisation und preemptives Multitasking bietet, scheidet MSDOS als Betriebssystem für Echtzeitanwendungen aus¹.

¹ Es soll aber darauf hingewiesen werden, daß Echtzeiterweiterungen, wie z.B. das Produkt RTKERNEL für MS-DOS existieren, mit denen sich Echtzeitanwendungen komfortabel realisieren lassen.

7 OS-9 Softwareumgebung bzw. Erweiterung , GNU-C, UNIX-Lib, Device-Treiber, CURSES usw.

Softwareentwicklungsumgebung unter OS-9:

Auf dem einzusetzenden VME-Bus-System läuft MICROWARE OS-9, erweitert um ein TCP/IP-Paket der Firma N.A.T., welches Telnet, FTP (File Transfer Protokoll) und NFS-Client- und Server-Funktionalitäten zur Verfügung stellt. Als grafische Benutzerumgebung dient eine eingeschränkte, an OS-9 angepaßte Version von X-Windows.

MICROWARE liefert mit OS-9 den MICROWARE C-Compiler, einen Linker und Binder, ein Make-Utility und den Source-Level-Debugger aus. Der Standard-Editor des Systems ist eine OS-9-Portierung des MICROEMACS. Zusätzlich steht mit dem ANSI-C kompatiblen ULTRA-C noch ein zweiter C-Compiler von Microware zur Verfügung, allerdings ist die vorhandene frühe ULTRA-C-Version noch in einigen Komponenten fehlerbehaftet. Aus diesem Grund stellte sich die Frage, ob nicht zusätzlich freie Software zur Entwicklung herangezogen werden sollte. Eine umfangreiche Recherche im Internet bzw. im Use-Net nach freier Software für OS-9 [OS-9FAQ97] führte zu den OS-9-Portierungen der GNU-Utilities.

7.1 GNU-Software

Beim GNU-Projekt (selbstbezügliche Definition von GNU: GNU is not UNIX) der Free Software Foundation handelt es sich um eine Sammlung hochwertiger, freier Software. Ziel der Free Software Foundation (FSF) ist es, ein Unix-kompatibles Betriebssystem zu entwickeln, daß nur aus freier Software besteht. Frei bedeutet im Sinne der FSF, daß das System der GNU General Public License (GPL) unterliegt, die unter anderem fordert, daß bei der Weitergabe der Software grundsätzlich auch der komplette Quelltext zur Verfügung gestellt werden muß. Die GNU-Programme zeichnen sich durch außergewöhnliche Portabilität und Funktionalität aus. Einige der wichtigsten GNU-Programme sind auf OS-9 portiert worden und stehen auf FTP -Servern zum Download zur Verfügung [FTP97].

Der GNU-C/C++-Compiler "gcc" ist wohl das bekannteste GNU-Produkt und stellt einen sehr portablen, leistungsfähigen und ausgezeichnet dokumentierten Compiler zur Verfügung. Die Portierung der Version 2.6 des GNU-Compilers auf OS-9 von Stefan Paschernag [PASCHERNAG93] ist der leistungsfähigste, momentan verfügbare C-Compiler für OS-9. Für dieses Projekt wurde der GNU-C-Compiler verwendet, weil er den vorhandenen MICROWARE-C-Compilern klar überlegen ist.

7.2 Die Curses-Bibliothek

Im Gegensatz zur ursprünglichen Implementation im C-Dialekt "BORLAND C" bietet weder der Sprachumfang des Kerninghan-Ritchie-, noch des GNU-ANSI- C-Dialekts komfortable Bildschirmsteuerungsfunktionen. Für verschiedene UNIX-Dialekte existiert die sogenannte "Curses-Bibliothek", die der geräteunabhängigen fensterorientierten Terminal- Ein-, und Ausgabe dient. Unterstützt wird ein Fensterkonzept, daß sowohl Überlagerungen von Fenstern, als auch die Ein- und Ausgabe von Zeichenketten auf dargestellte oder verdeckte Fenster ermöglicht. Durch eine Optimierung der Cursor-Bewegungen auf dem Terminal sind die Fenster-Ausgaben durch eine gute Performance gekennzeichnet.

Da eine portierte Version der "NCURSES"-Bibliothek (NEW CURSES), einer erweiterten Variante von Curses, für OS-9 zur Verfügung steht [OS-9CUR90], wurden die mächtigen Bildschirmsteuerungsfunktionen von BORLAND-C in der Implementation auf die Curses-Bibliothek abgebildet. Durch die Verwendung der Curses-Bibliothek entsteht ein hochportabler und geräteunabhängiger Programmcode.

7.3 Das OS-9LIB

Die Curses-Bibliothek benötigt ebenso wie andere Software, die von Unix nach OS-9 portiert wurde, das sogenannte OS-9LIB. OS-9LIB ist eine frei verfügbare Bibliothek, die geschaffen wurde, um Portierungen von Unix nach OS-9 zu erleichtern. Sie macht das UNIX-ähnliche OS-9 noch UNIX-ähnlicher und stellt so verschiedene UNIX-Funktionalitäten zur Verfügung für die es unter OS-9 keine äquivalenten Funktionen gibt, z.B. die Behandlung von UNIX-Signalen. Unix-kompatible C-Header-Dateien, die zum Umfang der OS-9LIB gehören, ermöglichen die einfache Übernahme von großen Teile von UNIX-C-Quellen.

8 Portierung auf OS-9

Allgemein erfordert eine Portierung von in C geschriebener Software auf ein anderes Zielsystem besondere Aufmerksamkeit in folgenden Punkten:

- ♦ Die Implementierungen der C-Compiler können sich hinsichtlich der Datentypen unterscheiden. Im Allgemeinen ist beispielsweise eine Integer-Zahl auf verschiedenen Hardwareplattformen unterschiedlich lang repräsentiert (2 Byte oder 4 Byte). Solche Probleme können entweder durch die Definition geeigneter hardwareunabhängiger Datentypen in den Header-Dateien umgangen, oder durch Anpassung der Datentypen auf das Zielsystem vermieden, werden.
- ♦ Das Big-Endian/Little-Endian-Problem taucht auf, weil unterschiedliche Prozessoren bei Wordoperationen (16Bit) die beiden einzelnen Bytes unterschiedlich speichern. Intel-Prozessoren speichern das niedrigersignifikante Byte (Low-Byte) an der Speicherstelle vor dem höhersignifikanten Byte (High-Byte), Motorola-Prozessoren handhaben dieses genau umgekehrt, zuerst das High-Byte dann das Low-Byte. Solange die Daten nicht in Bytes aufgeteilt werden, besteht bei der Portierung kein Problem, wenn sie aber in Bytes aufgeteilt an einen Fremdrechner gesendet werden, der das jeweils andere Format voraussetzt, müssen die Bytes getauscht werden. In der Portierung für OS-9 übernimmt dieses die neue Funktion :

```
short Short_Int_Byte_Swap(short short_integer_wert)
```

8.1 Die Input/Output-Routinen

Die ursprüngliche Implementation bedient sich einer Interrupt-Service-Routine (ISR) um serielle Daten zu empfangen und zu versenden. Unter OS-9 ist der für die seriellen Schnittstellen zuständige Device-Driver schon echtzeitfähig. Die Implementation einer ISR entfällt somit unter OS-9. Um möglichst viel Programmcode weiterverwenden zu können, wurden die unter OS-9 nicht vorhandenen Funktionen der ISR und spezifische BORLAND-C-Spracherweiterungen als neue Funktionen nachgebildet.

Funktionsname	Aufgabe in der ursprünglichen Implementierung	Neue OS-9-Funktion
void ClearPendingInterrupts (Comtype ComPort)	Wartet auf den Abschluß eines anstehenden Interrupts.	Unter OS-9 obsolet. Funktion als Dummy implementiert.
boolean V24DataAvail (ComType ComPort)	Überprüft, ob ganze Datenpakete im Device-Treiber-Puffer vorliegen.	Funktionalität unter OS-9 nachgebildet.

void V24IniBuf (ComType ComPort)	Device-Treiber-Puffer löschen.	Unter OS-9 mit Hilfe von OS-9-Systemaufrufen realisiert.
byte V24GetByte (ComType ComPort)	Diese Funktion holt ein Byte aus dem Ringpuffer. Es wird vorausgesetzt, daß durch V24DataAvail zuvor sichergestellt wurde, daß auch ein Zeichen zum Auslesen bereitsteht!	Unter OS-9 mit Hilfe von OS-9-Systemaufrufen realisiert.
void V24SendByte (ComType ComPort, BYTE Data)	Schreibt ein Byte auf den aktuellen Com-Port.	Unter OS-9 mit Hilfe von OS-9-Systemaufrufen realisiert.
void InitCom (ComType ComPort)	Initialisiert eine serielle Kommunikationsschnittstelle.	Unter OS-9 mit Hilfe von OS-9-Systemaufrufen realisiert.
void DisableCom (ComType ComPort)	Schließt eine serielle Schnittstelle für die Kommunikation.	Unter OS-9 mit Hilfe von OS-9-Systemaufrufen realisiert.
boolean V24BytesAvail (ComType ComPort)	Neue Funktion für OS-9.	Überprüft, ob einzelne Bytes im Device-Treiber-Puffer vorliegen. Verbessert die Test-Port-Funktion!
void V24ReceivePackets (ComType ComPort)	Neue Funktion für OS-9. Diese Funktion ist eine Abbildung der Funktionen V24Int_x (aus ISR) des Originalprogrammes.	Überprüft, ob Daten im Device-Treiber-Puffer vorliegen und versucht, daraus Pakete zu bilden.
void V24ReceiveKugelPacket (void)	Neue Funktion für OS-9. Diese Funktion ist eine Abbildung der Funktionen V24Int_x (aus ISR) des Originalprogrammes.	Überprüft, ob Daten im Device-Treiber-Puffer vorliegen und versucht, daraus Kugel-Pakete zu bilden. Für die DIMENSION6-Steuerkugel.
void V24ReceiveKugelPacket_neue_Kugel (void)	Neue Funktion für den ISRA-Teach-Ball.	Wie oben, jedoch für den ISRA-Teach-Ball.
void V24ReceiveVALPacket (void)	Neue Funktion für OS-9. Diese Funktion ist eine Abbildung der Funktionen V24Int_x (aus ISR) des Originalprogrammes.	Überprüft, ob Daten im Device-Treiber-Puffer vorliegen und versucht, daraus VAL-II-Pakete zu bilden.

Für die Realisierung dieser Funktionen wurden folgender OS-9-Systemaufrufe verwendet:

OS-9-Systemaufruf	Bedeutung
<code>int _gs_rdy(path_desc)</code>	Überprüft, wieviele Zeichen im Puffer des Device mit dem Descriptor <code>path_desc</code> vorliegen. Ist Null, wenn kein Zeichen vorliegt.

Für die Ein-Ausgaben auf die Seriellen Ports wurden die Standard-C-Funktionen `open()`, `read()`, `write` und `close()` verwendet.

8.2 Polling im Vergleich zu Sleep()

Weil die Interrupt-Service-Routine unter OS-9 nicht mehr erforderlich ist wird im laufenden ALTER-Betrieb nur noch die Hauptsteuerschleife des VAL-5-Programmes ausgeführt. In der ursprünglichen Implementierung werden hier die Datenquellen Robotersteuerung, Steuerkugel, Workstation und Tastatureingaben pollend¹ abgefragt, und Bildschirmausgaben getätigt.

Der pollende Betrieb widerspricht dem Konzept von Echtzeit- und Multitaskingprogrammierung, weil keine Rechenzeit an andere Prozesse bzw. Systemprozesse abgegeben wird.

Unter OS-9 läßt sich Polling einfach verhindern, indem ein auf ein Ereignis wartender Prozeß mit der Funktion `_ss_sig()` ein Signal definiert, durch welches er geweckt werden soll. Der Prozeß geht dann über den Systemaufruf `sleep()` in den schlafenden Zustand über, in dem er keine Rechenzeit mehr verbraucht. In der Implementation wird dieses Verhalten so initialisiert:

Codesequenz	Erklärung
<pre>sigmask(1); _ss_sig(0, 1); _ss_sig(OS9_path_desc[VALPORT], 1); _ss_sig(OS9_path_desc[KUGELPORT], 1); sleep(0); _ss_rel(0, 1); _ss_rel(OS9_path_desc[VALPORT], 1); _ss_rel(OS9_path_desc[KUGELPORT], 1);</pre>	<p>Signal-Maskierung. Signale müssen zwischen <code>_ss_sig()</code>- und <code>sleep()</code>-Aufruf maskiert werden.</p> <p>Signal für Standardinput, Kugelport und VALport werden gesetzt.</p> <p>Der Prozeß geht in den Zustand Schlafend.</p> <p>Signale für Standardinput, Kugelport und VALport werden wieder gelöscht.</p>

¹ Im pollenden Betrieb läuft ein Programm in einer Endlosschleife, in der es auf Eingaben wartet und keine Rechenzeit abgibt.

8.3 Die Bildschirmausgabe

In der ursprünglichen Implementation wurden Bildschirmausgaben während des zeitkritischen ALTER-Betriebes mit Hilfe einer schnellen Assemblerroutine direkt in das Video-RAM getätigt, da die BORLAND-Bildschirmroutinen für diese Aufgabe zu langsam sind.

Unter OS-9 stehen solche Möglichkeiten nicht direkt zur Verfügung, wie ein UNIX-Betriebssystem verwaltet OS-9 Ein-Ausgabe über Devices. Trotz Verwendung der schnellen Curses-Bibliothek muß die Ausgabe auf ein Terminal, daß entweder über eine serielle Schnittstelle angesteuert, oder in der Grafischen Benutzerumgebung X-Windows auf einem Grafikbildschirm emuliert wird, langsamer sein als daß direkte Schreiben von Zeichen im Textmodus in das Video-RAM.

Also ist es sinnvoll die Bildschirmausgaben über einen zweiten Prozeß zu realisieren. Da aber unter OS-9 die Standardzeitscheibe 10 ms beträgt, würde ein Taskwechsel zum Anzeigeprozess die nötige Reaktionszeit für den ALTER-Prozess überschreiten. Lösen läßt sich dieser Konflikt durch die Vergabe unterschiedlicher Prioritäten für die Prozesse. Der hochpriorisierte VAL-5-ALTER-Prozess bekommt immer Rechenzeit wenn er sie benötigt, wenn er aber auf externe Ereignisse wartet, gibt er Rechenzeit ab, wodurch der niedrigpriorisierte Anzeigeprozess zum Zuge kommt. In der derzeitigen Implementation wird der Anzeigeprozess bei Bedarf manuell gestartet und die Priorität des VAL-5-Prozesses manuell mit dem Shell-Kommando `setpr` erhöht.

8.3.1 Die BORLAND Ausgabefunktionen

Folgende Funktionen des BORLAND-C existieren nicht im GNU-C-ANSI Dialekt, und wurden aus diesem Grund mittels der Curses-Bibliothek nachgebildet :

Funktionsname	Argumente	Zweck
<code>void gotoxy(int x,int y)</code>	x : X-Koordinate im aktuellem Fenster y : Y-Koordinate im aktuellen Fenster	Gehe zur Pos. x-y im aktuellen Fenster
<code>void clrscr()</code>	keine	Lösche Bildschirm im aktuellen Fenster
<code>void clreol()</code>	keine	Lösche von der aktuellen Cursor-Position bis zum Ende der Zeile im aktuellen Fenster
<code>void cprintf(char *formstr, char *str)</code>	formstr : Formatstring; str : Ausgabestring	Ausgabe in ein Fenster

Folgende Funktionen des Ursprungsprogrammes verwendeten komplexe Fensterfunktionen von BORLAND-C und mußten deshalb komplett mit Hilfe von Curses-Bibliotheken neu implementiert werden:

<pre>void iniwin(int winnr, int ls, int lz, int rs, int rz, BOOLEAN rand, char *titel)</pre>	<pre>winnr : Nummer der Fenster; ls, lz, rs, rz : Fenster -koordinaten *titel : Fenstertitel rand : Rand erzeugen</pre>	<p>Initialisierung der globalen Fenster</p>
<pre>void cprintfwin(int winnr, int x, int y, char *formstr, char *str)</pre>	<pre>winnr : Nummer der Fenster; x,y : Spalte, Zeile; formstr: Formatstring; str : Ausgabestring</pre>	<p>Ausgabe eines formatierten Strings in ein vordefiniertes Fenster an der Stelle x,y</p>

8.3.2 Die Curses-Fensterfunktionen

Fenster oder "Windows" werden in der Curses-Bibliothek als zweidimensionale Zeichenarrays, die den Bildschirm oder Teile davon repräsentieren, behandelt. Das Standard-Fenster wird mit der Funktion `initscrn()`, welche immer am Anfang eines Curses-Programmes steht, initialisiert. Mit `newwin()` können weitere Fenster kreiert und über Zeigervariable vom Typ `WINDOW*` referenziert werden. Ausgaben in ein Fenster werden nicht sofort sichtbar, sondern erst mit der Funktion `refresh()` auf dem physikalischen Bildschirm abgebildet. Fenster können sich überlappen und können auf dem Bildschirm bewegt werden. Pads sind spezielle Fenster, die nicht durch die Bildschirmgröße begrenzt und nicht in ihrer vollen Größe sichtbar sein müssen. Bei der Ausgabe auf den physikalischen Bildschirm optimiert die Curses-Bibliothek den Ausgabedatenstrom. So wird beispielsweise bei einem Bildschirm-Update mit `refresh()` nicht der ganze Bildschirminhalt neu ausgegeben, sondern nur die durch Curses-Ausgabefunktionen veränderten Bildschirmteile. Die Funktionen `wnoutrefresh(win)` und `doupdate()` optimieren den Update mehrerer Fenster auf das Terminal, weil Curses zusätzlich zu den Window-Strukturen zwei weitere, einen physikalischen Screen (beschreibt den aktuellen Bildschirm) und einen virtuellen Screen (beschreibt den gewünschten Bildschirm) enthält. Die Funktion `wnoutrefresh(win)` kopiert das angegebene Fenster auf den virtuellen Screen, während `doupdate()` den virtuellen und den physikalischen Screen miteinander vergleicht und daraus den aktuellen Update generiert. Indem bei Ausgaben auf verschiedene Fenster die Funktion `wnoutrefresh(win)` für jedes Fenster einzeln aufgerufen wird, gefolgt von einem einzelnen `doupdate()`, erfolgt die Ausgabe in optimierter Form. [ILLIK90]

Um die BORLAND-C Funktionalitäten unter OS-9 nachzubilden wurden folgende Grundfunktionen der Curses-Bibliothek verwendet:

verwendete Curses Funktionen	Argumente	Zweck
<code>WINDOW *initscrn()</code>	keine	Initialisiert Terminal-Umgebungsvariablen

<code>int endwin()</code>	keine	Ursprünglichen Terminalzustand wiederherstellen
<code>WINDOW *newwin(int nlines, int ncols, int begin_y, int begin_x)</code>	<p><code>nlines</code> : Anzahl der Linien</p> <p><code>ncols</code> : Anzahl der Spalten</p> <p><code>begin_y</code> : y-Koordinate des ersten Fensterpunktes</p> <p><code>begin_x</code> : x-Koordinate des ersten Fensterpunktes</p>	Kreiert neues Fenster
<code>int delwin(WINDOW *win)</code>	<code>win</code> : Zeiger auf ein Fenster	Löscht ein Fenster
<code>int wclrtoeol(WINDOW *win)</code>	<code>win</code> : Zeiger auf ein Fenster	Löscht Fenster ab aktueller Bildschirmposition bis zum Zeilenende
<code>int box(WINDOW *win, chtype vert, chtype hor)</code>	<p><code>win</code> : Zeiger auf ein Fenster</p> <p><code>vert, hor</code>: Zeichen für horizontale und vertikale Umrahmung</p>	Umrahmt ein Fenster
<code>int wprintw(WINDOW *win, const char *formatstr, ...)</code>	<code>win</code> : Zeiger auf ein Fenster	Wirkung und Formatangaben wie <code>printf()</code> der Standard-C-Bibliothek
<code>int mvwprintw(int y, int x, WINDOW *win, const char *formatstr, ...)</code>	<code>win</code> : Zeiger auf ein Fenster	Wirkung und Formatangaben wie <code>printf()</code> der Standard-C-Bibliothek, Ausgabe an Position <code>x, y</code> des aktuellen Fensters
<code>int wnoutrefresh(WINDOW *win)</code>	<code>win</code> : Zeiger auf ein Fenster	Kopiert das angegebene Fenster auf den virtuellen Screen
<code>int doupdate()</code>	keine	Generiert den aktuellen Update des virtuellen auf den physikalischen Bildschirm
<code>int refresh()</code>	keine	Ausgabe auf Terminal ausführen

Eine Besonderheit stellt die unter BORLAND-C zur Verfügung gestellte Funktion `kbhit()` dar. Die Abfrage des Tastaturpuffers ohne Programmunterbrechung für die Eingabe ist eine systemnahe Funktion, die in keinem Standard-C-Sprachumfang definiert ist und deshalb immer durch Betriebssystemspezifische Aufrufe bzw. hardwarenahe Programmierung realisiert wird. OS-9 realisiert solche Abfragen über die Betriebssystemfunktion `_gs_rdy(0)` welche EOF als Returnwert zurück liefert, wenn sich kein Zeichen im Tastaturpuffer bzw. in `stdin` (Standard-Input-Device) befindet.

9 Erweiterung der Funktionalität gegenüber der ursprünglichen Implementierung

9.1 Debug-Funktion zur Protokollanalyse

Um bei Abbruch des ALTER-Betriebs eine Möglichkeit der Protokollanalyse zu schaffen, wurde zusätzlich eine Debug-Funktion zur Protokollanalyse implementiert. Sie wird im Menü "Optionen" über den Untermenüpunkt "I/O-Debug" aufgerufen. Mit Hilfe dieser Funktion lassen sich jeweils die letzten 1000 gesendeten und empfangenen Bytes analysieren. Außerdem erlaubt diese Funktion die Ausgabe von aktuellen ALTER-Daten für die Analyse des verbesserten kumulativen ALTER-Betriebes.

Zusätzlich sind im Menü "Optionen" die Untermenüpunkte "Anschlußbelegung" und "Kugeltyp", zur Anzeige der aktuellen Anschlußbelegung der seriellen Schnittstellen und zur Auswahl der verwendeten Steuerkugel, eingebunden worden.

9.2 Verbessertes kumulatives ALTER-Betrieb

Der kumulative ALTER-Betrieb in der ursprünglichen Implementierung erwies sich als nicht ruckfrei. Wenn die Steuerkugel plötzlich bewegt oder losgelassen wurde, mußte der Roboter im ALTER-Betrieb sofort stoppen bzw. anfahren, was zu ruckenden für die Roboterachsen schädlichen Bewegungen führte. Es bestand der Bedarf nach einem Algorithmus, welcher zu heftige Bewegungsänderungen pro Zeiteinheit (also Beschleunigungen) vermeidet. Zu diesem Zweck wurde die Funktion `Kugel_Data_Rcvd()` dahingehend modifiziert, daß zusätzlich zu einer Geschwindigkeitsbegrenzung eine Beschleunigungsbegrenzung implementiert wurde. Für jede Koordinate in translatorischer- und rotatorischer Richtung wird eine maximale Änderung des ALTER-Wertes pro Kommunikationszyklus zugelassen. Diese maximale Änderung wird auch im abbremsenden Betrieb berücksichtigt, so daß in keinem Fall höhere Beschleunigungen auftreten.

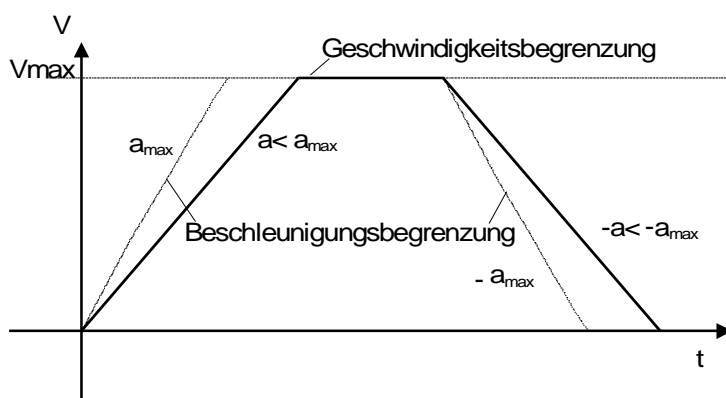


Bild 9. Geschwindigkeits- und Beschleunigungsbegrenzung im ALTER-Betrieb

9.3 Integration einer "Clamp"-Funktion

Im Laufe der Portierung entstand der Bedarf, zusätzlich den Greifer des Roboters über die Funktionstasten der Steuerkugel bedienen zu können. Der Greifer des Roboters wird von der Robotersteuerung durch den Befehl `CLOSEI` geschlossen und durch `OPENI` wieder geöffnet. Er wird pneumatisch betätigt und kennt nur die beiden Zustände "offen" und "geschlossen".

Wie bereits in 4.2 erläutert, übergibt die Steuerkugel den Zustand der acht Funktionstasten in einem Byte an den Steuerrechner, wobei jedes Bit eine Taste repräsentiert. Das `VAL_5`-Programm auf dem Steuerrechner wertet nun die Funktionstastenzustände aus und sendet bei Bedarf an die Robotersteuerung einen `ALTER`-Ausnahmecode, wie in 3.3.2 beschrieben.

Der neue `ALTER`-Befehl zum Start des `ALTER`-Betriebs enthält nun den Namen einer Subroutine, zu der verzweigt werden muß, wenn ein Ausnahmecode übermittelt wird.

Die `VAL`-Programme für das neue Verhalten haben folgenden Inhalt:

Programm "apollo":

```

1          SPEED 5
2          MOVE #apollo
3          ALTER (0, 22, clamp2, 127)
4      10   MOVES HERE
5
6          GOTO 10

```

Programm "clamp2":

```

1          TYPE ERROR(-1)
2          test = ERROR(-1)
3          IF test == 253 GOTO 10
4          IF test == 254 GOTO 20
5          GOTO 100
6      10   CLOSEI
7          GOTO 100
8      20   OPENI
9          GOTO 100
10     100  RETURN

```

Der vom Steuerrechner gesendete Ausnahmecode wird in `VAL_II` in der Variablen `ERROR(-1)` ausgewertet, und führt so zur Abarbeitung des `CLOSEI`- oder `OPENI`-Befehls.

9.4 Verbessertes Handling der Grafik-Workstation

Ein Fehlerverhalten trat in der ursprünglichen Implementation auf, wenn bei aktivierter Simulation unter SMS der ALTER-Betrieb auf dem externen Steuerrechner eingestellt worden war. Wenn die Simulation auf der Workstation nicht umgehend beendet wurde, sendete die APOLLO weiterhin jeden Darstellungszyklus ein Enquire (ENQ) an den Steuerrechner, dieser war nach Beendigung des ALTER-Betriebes nicht mehr in der Lage das Enquire zu empfangen. Weil die Versendung von Zeichen über die serielle Schnittstelle im UNIX-Dialekt DOMAIN OS device-orientiert verläuft, wird jedes ENQ, das nicht gesendet werden kann, im Sendepuffer aufbewahrt, bis wieder eine Verbindung besteht. Bei erneuter Aufnahme des ALTER-Betriebes durch den externen Steuerrechner, nimmt dieser auch den Handshakebetrieb zur Simulationsumgebung wieder auf. Der Steuerrechner empfängt jetzt in schneller Folge ENQ-Zeichen und antwortet dem Protokoll entsprechend mit ebenso vielen Datenpaketen, was auf Apollo-Seite zu einem Pufferüberlauf und den Absturz der Simulationsumgebung führt. Dieses Verhalten ließ sich nur vermeiden, wenn die Simulation vor dem ALTER-Betrieb angehalten wurde, oder die Simulationsumgebung prophylaktisch verlassen und neu gestartet¹ wurde, bevor der ALTER-Betrieb wieder aufgenommen werden sollte.

Für die neue Implementation wurde das Verhalten der Workstation so verändert, daß außer beim ersten Aufruf ein ENQ immer nur dann gesendet wird, wenn vorher ein Nachrichtenpaket vom externen Steuerrechner empfangen wurde. Aber auch bei diesem Verhalten ist noch der Fall zu berücksichtigen, daß der ALTER-Betrieb gerade in dem Augenblick beendet wird, in dem bereits ein ENQ an den externen Steuerrechner gesendet wurde, dieser aber nun nicht mehr antworten kann. Weil die zur Verfügung stehende Softwareschnittstelle keine einfache Möglichkeit der Benutzerinteraktion bietet, wird ein erneutes Versenden eines ENQ-Zeichens durch kurzes Unterbrechen (ca. zwei Sekunden) des Simulationsablaufes mittels eines Mausclicks auf das Startsymbol initiiert. Diese Veränderungen wurden in der Funktion "puma_koll_verm.c" vorgenommen, die bei angewähltem Startsymbol in jedem Simulationszyklus, nicht aber bei einer Unterbrechung, durchlaufen wird. Mittels Abfrage der Systemzeit erkennt die neue Programmsequenz, ob das Startsymbol kurz abgeschaltet war und sendet dann ein neues ENQ-Zeichen.

Mit diesen Modifikationen konnte ein stabiles Betriebsverhalten in der Kommunikation zwischen Visualisierungssystem und Steuerrechner erreicht werden.

9.5 Einbindung der Kommunikation mit dem ISRA-Robot-Teach-Ball in den Steuerfluß von VAL5

Da der ISRA-Robot-Teach-Ball niemals Daten ohne Aufforderung an den Steuerrechner sendet, mußte der Steuerfluß in der Implementation entsprechend angepaßt werden. Die Sendung eines ENQ-Zeichen erfolgt jetzt zeitgesteuert.

Die neue Implementation enthält zusätzlich eine neue Testroutine für den ISRA-Robot-Command-Teach-Ball.

¹ Dieser Vorgang dauert immer etwa zwei Minuten.

10 Inbetriebnahmeanleitung

Diese Inbetriebnahmeanleitung stellt ein kurzes Bedienhandbuch und eine Dokumentation der Verkabelung der involvierten Komponenten dar. Sie ersetzt nicht die Dokumentation der einzelnen Komponenten.

10.1 Verkabelung

Für die Inbetriebnahme der Virtual-Reality-Umgebung ist folgende Verkabelung des Steuerrechners nötig:

Anschluß	OS-9-Device-name	Kabelnummer	Verbindung mit
2	/t1	1 und 2	Robotersteuerung (über RS232-RS422 Umsetzer an Anschluß J123)
3	/t2	3	Steuerkugel ISRA-Robot- Command-Teach- Ball
4	/t3	4	Apollo-Workstation SIO 2

10.2 Kabeldokumentation

Die angefertigten Kabel sind folgendermaßen verdrahtet:

Kabel 1:

Steuerrechner MICROSYS		RS-232-RS-422-Umsetzer RS-232-Seite	
9-Pol. D-Sub weiblich		25-Pol. D-Sub weiblich	
Signal	Pin-Nummer	Pin-Nummer	Signal
RXD Recieve Data	2	3	TXD Transmit Data
TXD Transmit Data	3	2	RXD Recieve Data
CTS Clear to Send	8	5	RTS Request to Send
GND System Ground	5	7	GND System Ground

Kabel 2:

RS-232-RS-422-Umsetzer RS-422-Seite		Robotersteuerung	
25 Pol. D-Sub männlich		10 Pol. Spezialstecker männlich	
Signal	Pin- Nummer	Pin- Nummer	Signal
DOut A	10	A	Receive+
DIn B	11	F	Transmit+
DOut B	22	B	Tranceive-
DIn B	23	D	Receive-

Kabel 3:

Steuerrechner MICROSYS		Workstation APOLLO	
9-Pol. D-Sub weiblich		25-Pol. D-Sub männlich	
Signal	Pin- Nummer	Pin- Nummer	Signal
RXD Recieve Data	2	2	RXD Receive Data
TXD Transmit Data	3	3	TXD Transmit Data
GND System Ground	5	7	GND System Ground

Kabel 4:

Steuerrechner MICROSYS		ISRA-Robot-Command-Teach- Ball	
9-Pol. D-Sub weiblich		9-Pol. D-Sub weiblich	
Signal	Pin- Nummer	Pin- Nummer	Signal
RXD Recieve Data	2	3	TXD Transmit Data
TXD Transmit Data	3	2	RXD Receive Data
GND System Ground	5	7	GND System Ground

Alle Kabel sind hier von Endgerät zu Endgerät beschrieben¹.

10.3 Vorgehensweise am Steuerrechner (VME-Bus-System)

- ◆ ISRA-Steuerkugel einschalten.

¹ Die an der MICROSYS angeschlossenen 9 Pol. D-Sub auf 25 Pol. D-Sub Umsetzer sind Nullmodemkabel. (Tauschen Leitung 2 und 3 usw.)

- ◆ Anmelden als Benutzer "ic_40".
- ◆ Am Prompt den internen CPU-Cache mit "cache_on" einschalten.
(Auf dem VME-Bus-System läuft Software die erfordert, daß der CPU-Cache abgeschaltet ist.)
- ◆ Am Prompt den Befehl "setenv TERM ansi" ausführen, Das Curses-Library erfährt aus dieser Environmentvariable der verwendeten Terminaltyp.
- ◆ Am Prompt "start_demo" ausführen.
- ◆ Evtl. Menüpunkt "Kugelttest" ausführen, um korrekten Betrieb der Steuerkugel zu überprüfen
- ◆ Bei Bedarf Anzeigeprozess starten:
 - Im Menü "Optionen" "Koproz." anwählen um die Kommunikationsadressen für
commstat und norm_nachr_von zu erfahren.
 - In einem zweiten Terminalfenster am Prompt "val_5_ausgabe" ausführen und
dort die Kommunikationsadressen eingeben.
 - In einem dritten Terminalfenster die Priorität von VAL_5 auf 500 setzen
(geht mit "setpr [Prozeßnummer] 500" , Prozeßnummer mit "procs"
feststellen)
- ◆ ALTER-Betrieb mit Taste "A" starten.
- ◆ ESC drücken um den ALTER-Betrieb zu beenden.

10.4 Vorgehensweise an der Robotersteuerung

- ◆ Hauptschalter der Robotersteuerung einschalten.
- ◆ Systemterminal in Betrieb nehmen.
- ◆ Mit Schlüsselschalter den eingebauten Rechner in Betrieb nehmen.
- ◆ Der eingebaute Rechner verfügt über ein CMOS-RAM, deshalb die Fragen
"Load VAL from Floppy (Y/N)"
und "Initialisize (Y/N)" mit N beantworten.
- ◆ Die Roboter-"Arm-Power" einschalten.
- ◆ Am Prompt den Befehl "cal" ausführen, der Roboter wird kalibriert.
- ◆ Am Prompt den Befehl "do move #apollo" ausführen, der Roboter fährt an die Startposition #apollo.
- ◆ Am Prompt den Befehl "ex apollo" ausführen,
nachdem am externen Steuerrechner der ALTER-Betrieb gestartet wurde.
- ◆ der ALTER-Betrieb wird gestartet.

Das Steuerprogramm apollo enthält folgendes kurzes Roboterprogramm:

```
SPEED 10
ALTER(0,23)
10 MOVES HERE
GOTO 10
```

10.5 Vorgehensweise an der Apollo-Workstation

- ◆ Anmelden am System unter dem Account "bischoff".
- ◆ Am Prompt mit "wd sms/sms_3.0x" in das SMS-Verzeichnis wechseln.

- ◆ Am Prompt "sms" eingeben um SMS zu starten.
- ◆ Den Menüpunkt "SIRPA" anwählen.
- ◆ Die Umgebung "work_neu.sirpa" anwählen und "übernehmen" anklicken.
- ◆ Menüpunkt "Programmerstellung" anwählen.
- ◆ Im Menüpunkt "Kollisionsvermeidung" "Kollisionsvermeidung Ein" anwählen.
- ◆ Im Menüpunkt "Parameter" "Kontinuierlich" anwählen.
- ◆ Menüpunkt "Start" anwählen und mit gedrückter Maustaste verlassen.
 Wenn die Simulation hängenbleibt, etwa durch zwischenzeitliche Beendigung des ALTER-Betriebes, einfach "Start" für etwa 5 Sekunden deaktivieren und wieder neu starten.

10.6 Bedienoberfläche von VAL_5

Das portierte OS-9-Programm VAL_5 unterscheidet sich in der Bedienung von der Originalversion nur in wenigen Punkten. Zusätzliche Funktionalitäten wurden im Menü Optionen untergebracht. Nach dem Aufruf von VAL_5 erscheint folgende Bedienoberfläche:

```

|-----|-----|-----|-----|
|VAL-Altermodus|VAL-Anfangsdaten|EXT-Systemmessg|
|              |              |PORT:/t1 geoeffnet|
|              |              |PORT:/t2 geoeffnet|
|              |              |PORT:/t3 geoeffnet| |
|---|---|---|---|
|VAL-Altb|VAL-Ctrlb|VAL-Segmentinfo|VAL-Transformation|
|         |         |Status   Nr. Move|
|-----|-----|-----|-----|
|Kugeldaten|
|-----|-----|-----|-----|
|CMD|
|(W)Stst (R)ugelst (T)estport (A)lterbetr (U)erweigern (O)ptionen (Q)uit >
|-----|-----|-----|-----|
    
```

Die einzelnen Fenster haben folgende Bedeutung:

VAL-Altermodus	Der aktuelle ALTER-Modus wird hier angezeigt
EXT-Systemmessg	System- und Statusmeldungen werden hier ausgegeben
VAL-Altb (*)	Die Auswertung des aktuellen Controllbytes für den ALTER-Modus (siehe Kap.3.2.2)
VAL-Ctrlb (*)	Die Auswertung des aktuellen Controllbytes für den ALTER-Kommunikationsstatus (siehe Kap.3.2.2)

VAL-Segmentinfo	Für Debugging-Zwecke
VAL-Transformation (*)	Die aktuelle empfangene Transformation wird hier ausgegeben
Kugeldaten	Fenster für Debugging der Kugelkommunikation
CMD	Kommandofenster, Ausgabe der Menüpunkte und Eingaben

Ausgabe in die mit (*) bezeichneten Fenster sind zeitkritisch und werden in dieser Implementation durch einen Koprozeß übernommen.

Diese Kommandos können angewählt werden:

(W)STst	Testet die Verbindung zur Workstation
(K)ugeltst	Testet die Steuerkugel
(T)estport	Testet Ports mit verschiedenen Baudraten
(A)lterbetr	ALTER-Betrieb wird gestartet. Kann mit Escape (ESC) beendet werden.
(V)erweigern	Verweigert den ALTER-Betrieb bei Verbindungsaufforderung von der Robotersteuerung
(O)ptionen	Ruft Untermenü Optionen auf: (1)Anschl.bel.: Anschlußbelegung wird ausgegeben (2)Kugeltyp einst.: Der Kugeltyp kann gewechselt werden (ISRA oder DIMENSION6) (3)IO-DEBUG: Verschiedene Kommunikationspuffer können eingesehen werden (4)Koproz.: Einstellungen zum Starten des Anzeigeprozesses (5)Clamp on/off: Clamp-Option kann hier Ein- und Ausgeschaltet werden
(Q)uit	Beendet das Programm

11 Beurteilung der erzielten Funktionalitäten

Durch die eingesetzten zusätzlichen Bibliotheken konnte in relativ kurzer Zeit die gesamte Funktionalität des Ursprungsprogrammes auf das Echtzeitbetriebssystem OS-9 übertragen werden. Außerdem gelang eine Verbesserung der Funktionalität gegenüber dem Ursprungsprogramm. Der entstandene Programmcode ist insgesamt portabler geworden, und würde so weitere mögliche Portierungen vereinfachen. Für das gestellte Problem erweist sich die zur Verfügung stehende Rechenleistung für einen Echtzeitbetrieb als ausreichend.

12 Ausblick

Es ist bereits vorgesehen die im VME-Bus-System vorhandenen Bildverarbeitungskarten künftig in das Virtual-Reality-Konzept zu integrieren, beispielsweise wäre eine Darstellung von im Arbeitsraum des Roboters real auftretenden Hindernissen in der Simulationsumgebung denkbar.

12.1 Teleoperation über das Internet

Eine ganz andere denkbare Weiterentwicklung der Teleoperation wäre die Möglichkeit, das VME-Bussystem über das Internet fernzusteuern. Es ist der einfach zu realisierende Fall denkbar, in dem die Apollo-Workstation mit beiden Steuerkugeln entfernt aufgestellt wird, und die Kommunikation mit dem VME-Bussystem über das TCP/IP-Protokoll abläuft.

Eine reizvolle Weiterentwicklung wäre die Ersetzung der Grafik-Workstation durch ein VRML-System, welches ein hardwareunabhängiges Konzept auf Client-Seite bereitstellen würde. Die Steuerkugel könnte in diesem Konzept durch ein allgemein mit einer Maus bedienbares Java-Steuer-Panel ersetzt werden. Um eine Echtzeitsteuerung zu ermöglichen ist allerdings eine gewisse garantierte Übertragungsbandbreite nötig, die allerdings um einige Zehnerpotenzen geringer liegen würde, als eine Lösung mit Echtzeitbildübertragung.

Die Entwicklung von VRML schreitet augenblicklich sehr dynamisch voran, so daß entsprechend leistungsfähige Client-Software schon bald existieren wird.

13 Literatur

- [CT1/95] C'T Magazin für Computertechnik, Ausgabe 1/95, Heise Verlag
[DICKEN92] Diplomarbeit 1992 am Lehrstuhl Prozeß-Steuerungs- und
Regelungstechnik an der Fernuniversität Hagen
[HOF88] Dipl.-Ing. Stefan Hofmann, Benutzerhandbuch für den
ISRA Robot Command & Teach Ball
[FTP97] FTP-Server für OS-9-Software <ftp://os-9archive.rtsi.com/OS-9>
[OS-9FAQ97] Die OS-9 Usenet FAQ (Frequently Asked Question)
<ftp://os-9archive.rtsi.com/OS-9/faq/os-9faq.html>
[OS-9CUR90] Curses-Library und C-Header für OS9, auf dem OS-9 FTP-Server
<ftp://os-9archive.rtsi.com/OS-9>
[UNIMATION] Benutzerleitfaden für VAL II , UNIMATION, Frankfurt
[ILLIK90] J. Anton Illik, Programmieren in C unter UNIX, Sybex 1990

Microware Systemhandbücher:

OS-9/68000 Operating System Technical Manual 1986/89

Using Professional OS-9

USING OS-9/INTERNET

OS-9 Assembler/Linker/Debugger Manual

Using μ macs

OS-9/68000 C Compiler User's Manual

OS-9/68000 Advanced System Software : OS-9/68000 Source Level Debugger User Manual

14 Anhang